Scalable and Efficient Uncertainty Estimation and Reduction in Edge-AI Accelerators

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Soyed Tuhin Ahmed
aus Sylhet (Bangladesh)

Tag der mündlichen Prüfung:	15 July 2024
1. Referent:	Prof. Dr. Mehdi B. Tahoori, Karlsruhe Institute für Technology (KIT)
2. Korreferent:	Prof. Dr. Krishnendu Chakrabarty, Arizona State University (ASU)

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Dr. Mehdi B. Tahoori. I am grateful for allowing me to work on exciting projects, giving me enough freedom to explore and define my own solutions, and for his guidance, insightful feedback, and constant availability for discussions. His mentorship not only taught me how to conduct meaningful research but also helped steer my work in the right direction.

I would also like to extend my sincere thanks to Prof. Dr. Lorena Anghel, Dr. Guillaume Prenat, and Kamal Danouchi from the University Grenoble-Alpes, CEA, and Spintec Lab, France. Their involvement, scientific expertise, and contributions to hardware implementation significantly enhanced the depth and practicality of my research. I am particularly grateful for their continuous support beyond the projects, including their invaluable assistance in preparing for my oral exams.

I am deeply appreciative of all my coauthors and colleagues for their collaboration and contributions. In particular, I would like to thank Dr. Michael Hefenbrock and Dr. Christopher Münch, who mentored me during the initial stages of my Ph.D. Their support and encouragement were instrumental in shaping my research trajectory, and their mentorship has been a constant source of inspiration.

Finally, I want to thank my family. Their help goes beyond this thesis. Throughout my academic and professional journey, I have been fortunate to receive support that ranges from financial assistance to life advice. Their unwavering support has been the foundation of my academic and professional career. I am deeply grateful for their love and support.

"Living on the edge is nice, but uncertainties come with it."

Abstract

In recent years, advancements in artificial intelligence (AI) have led to the widespread deployment of neural networks (NNs) in various domains, such as computer vision, speech recognition, and natural language processing. However, performance improvement comes at the cost of a substantial number of parameters and computation requirements. Thus, the deployment of AI applications predominantly relies on cloud computing. However, NN inference in the cloud is impractical for real-time and delay-sensitive edge applications due to low latency, bandwidth limitations, and privacy concerns. Consequently, edge AI, which involves deploying AI algorithms on edge devices, offers a solution by processing incoming data for inference on-device, thus reducing data communication latency and improving privacy.

Nevertheless, on-device inference is challenging since the inference stage of NN is computationally intensive, power-hungry, and has a high memory overhead. However, edge devices are typically resource-constrained in terms of computational and memory resources and are battery-powered. Therefore, performing inference on specialized AI accelerator architectures is an attractive solution that offers parallel processing, low-power consumption, reduced latency, and, in some architectures, in-memory computation.

Despite the benefits of edge AI accelerators, due to their deployment in dynamic and real-world environments, they are susceptible to various sources of uncertainty. For example, data distribution shifts and sensor noise can corrupt or perturb inference data. Despite the NN receiving such "out-of-distribution" data, they tend to make overconfident predictions. Similarly, hardware non-idealities such as defects and variations arising during the manufacturing process and online operation due to inherent device properties, immature fabrication processes, and external environmental factors such as temperature and radiation particles can also perturb parameters, activations, and outputs of edge AI accelerators. Consequently, these non-idealities negatively impact inference accuracy, significantly affect reliability, and introduce uncertainty in the functionality of the edge AI accelerator. Uncertain predictions are unacceptable for many applications, including safety-critical applications, where the consequences of an incorrect prediction can be catastrophic. Therefore, estimating uncertainty in both data and functionality, followed by uncertainty reduction, is crucial for ensuring the highly reliable operation of edge AI accelerators.

Existing methods often focus on specific aspects, such as fault tolerance on AI accelerators with specific memory technology or architecture, and suffer from resource scalability issues. Specifically, resources such as the number of random number generators (RNGs), dropout modules, models, test vectors, and memory increase with the size of the model, dataset, and other factors such as ensemble members. Therefore, they are impractical for resource-constrained edge AI accelerators. Nevertheless, the efficient implementation of certain methods, such as Bayesian Neural Networks, which are highly effective for predictive uncertainty estimation, is itself challenging and requires holistic algorithm-hardware co-optimization.

To address the limitations, this thesis makes several contributions: a) predictive uncertainty estimation, b) functional uncertainty estimation of edge AI accelerators, c) reducing uncertainty due to hardware non-idealities, and d) continuous availability of AI accelerators using algorithm-hardware co-design-based solutions. Specifically, problem-aware training algorithms, loss and penalty functions, model compression, novel neural network topologies, normalization layers, dropout methods, hardware architectures, parameter mapping, approximation methods, compact automatic test vector generation methods, and several online testing frameworks are proposed for our objective. The main goal is to improve the uncertainty quantification, testability, reliability, performance, manufacturing yield, and efficiency of edge AI in a resource-scalable manner. Extensive evaluations are performed for each method to demonstrate their

effectiveness. The contributions enhance the overall reliability and efficiency of Edge AI accelerators from a holistic perspective. Consequently, it enables the wide-scale deployment of NN in real-world applications.

Zusammenfassung

In den letzten Jahren haben Fortschritte im Bereich der künstlichen Intelligenz (KI) zu einem weit verbreiteten Einsatz neuronaler Netze (NN) in verschiedenen Bereichen geführt, z. B. in der Computer Vision, der Spracherkennung und der Verarbeitung natürlicher Sprache. Die Leistungsverbesserung geht jedoch auf Kosten einer beträchtlichen Anzahl von Parametern und Berechnungsanforderungen. Daher werden KI-Anwendungen überwiegend in der Cloud eingesetzt. Die NN-Inferenz in der Cloud ist jedoch für Echtzeit- und verzögerungsempfindliche Edge-Anwendungen aufgrund geringer Latenzzeiten, Bandbreitenbeschränkungen und Datenschutzbedenken unpraktisch. Folglich bietet die Edge-KI, bei der KI-Algorithmen auf Edge-Geräten eingesetzt werden, eine Lösung, indem eingehende Daten für die Inferenz auf dem Gerät verarbeitet werden, wodurch die Latenzzeit bei der Datenkommunikation verringert und der Datenschutz verbessert wird.

Dennoch ist die Inferenz auf dem Gerät eine Herausforderung, da die Inferenzphase der NN rechenintensiv und stromhungrig ist und einen hohen Speicherbedarf hat. Edge-Geräte verfügen jedoch in der Regel nur über begrenzte Ressourcen in Form von Rechen- und Speicherressourcen und sind batteriebetrieben. Daher ist die Durchführung von Schlussfolgerungen auf spezialisierten KI-Beschleunigerarchitekturen eine attraktive Lösung, die parallele Verarbeitung, geringen Stromverbrauch, geringere Latenzzeiten und bei einigen Architekturen auch In-Memory-Berechnungen bietet.

Trotz der Vorteile von Edge-KI-Beschleunigern sind diese aufgrund ihres Einsatzes in dynamischen und realen Umgebungen anfällig für verschiedene Unsicherheitsfaktoren. So können beispielsweise Verschiebungen in der Datenverteilung und Sensorrauschen die Inferenzdaten verfälschen oder stören. Obwohl die NN solche „verteilungsfremden“ Daten erhalten, neigen sie dazu, überhöhte Vorhersagen zu treffen. In ähnlicher Weise können auch Hardware-Nicht-Idealitäten wie Defekte und Schwankungen, die während des Herstellungsprozesses und des Online-Betriebs aufgrund von inhärenten Geräteeigenschaften, unausgereiften Herstellungsprozessen und externen Umweltfaktoren wie Temperatur und Strahlungsteilchen auftreten, die Parameter, Aktivierungen und Ausgaben von Edge-KI-Beschleunigern stören. Folglich wirken sich diese Nicht-Idealitäten negativ auf die Genauigkeit der Schlussfolgerungen aus, beeinträchtigen die Zuverlässigkeit erheblich und führen zu Unsicherheiten in der Funktionalität des Edge-KI-Beschleunigers. Unsichere Vorhersagen sind für viele Anwendungen inakzeptabel, einschließlich sicherheitskritischer Anwendungen, bei denen die Folgen einer falschen Vorhersage katastrophal sein können. Daher ist die Abschätzung der Unsicherheit sowohl bei den Daten als auch bei der Funktionalität und die anschließende Reduzierung der Unsicherheit entscheidend für die Gewährleistung eines äußerst zuverlässigen Betriebs von Edge-KI-Beschleunigern.

Bestehende Methoden konzentrieren sich oft auf spezifische Aspekte, wie Fehlertoleranz auf KI-Beschleunigern mit spezieller Speichertechnologie oder -architektur, und leiden unter Problemen der Skalierbarkeit von Ressourcen. Insbesondere nehmen Ressourcen wie die Anzahl der Zufallszahlengeneratoren (RNGs), Dropout-Module, Modelle, Testvektoren und Speicher mit der Größe des Modells, des Datensatzes und anderer Faktoren wie Ensemblemitglieder zu. Daher sind sie für ressourcenbeschränkte Edge-AI-Beschleuniger unpraktisch. Nichtsdestotrotz ist die effiziente Implementierung bestimmter Methoden, wie Bayes'sche Neuronale Netze, die für prädiktive Unsicherheitsabschätzungen sehr effektiv sind, selbst eine Herausforderung und erfordert eine ganzheitliche Algorithmus-Hardware-Ko-Optimierung.

Um diese Einschränkungen zu beheben, leistet diese Arbeit mehrere Beiträge: a) prädiktive Unsicherheitsabschätzung, b) funktionale Unsicherheitsabschätzung von Edge-KI-Beschleunigern, c) Reduzierung der Unsicherheit aufgrund von Hardware-Nicht-Idealitäten und d) kontinuierliche Verfügbarkeit von

KI-Beschleunigern durch Algorithmus-Hardware-Co-Design-basierte Lösungen. Konkret werden problemorientierte Trainingsalgorithmen, Verlust- und Straffunktionen, Modellkomprimierung, neuartige neuronale Netztopologien, Normalisierungsschichten, Dropout-Methoden, Hardwarearchitekturen, Parameter-Mapping, Approximationsmethoden, kompakte automatische Testvektor-Generierungsmethoden und verschiedene Online-Test-Frameworks für unser Ziel vorgeschlagen. Das Hauptziel besteht darin, die Quantifizierung der Unsicherheit, die Testbarkeit, die Zuverlässigkeit, die Leistung, die Produktionsausbeute und die Effizienz von Edge AI auf eine ressourcenskalierbare Weise zu verbessern. Für jede Methode werden ausführliche Bewertungen durchgeführt, um ihre Wirksamkeit zu demonstrieren. Die Beiträge verbessern die allgemeine Zuverlässigkeit und Effizienz von Edge-KI-Beschleunigern aus einer ganzheitlichen Perspektive. Folglich ermöglichen sie den breiten Einsatz von NN in realen Anwendungen.

List of Publications

The following list enumerates conference and journal papers by the author of this dissertation while pursuing his doctorate.

First-author conference papers that are part of this thesis

[1] Soyed Tuhin Ahmed, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "NeuroScrub: Mitigating retention failures using approximate scrubbing in Neuromorphic fabric based on resistive memories". In: *2021 IEEE European Test Symposium (ETS)*. IEEE. 2021, pp. 1–6.

[2] Soyed Tuhin Ahmed and Mehdi B Tahoori. "Fault-tolerant Neuromorphic Computing with Functional ATPG for Post-manufacturing Re-calibration". In: *2022 IEEE 40th VLSI Test Symposium (VTS)*. IEEE. 2022, pp. 1–7.

[3] Soyed Tuhin Ahmed, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "Process and Runtime Variation Robustness for Spintronic-Based Neuromorphic Fabric". In: *2022 IEEE European Test Symposium (ETS)*. IEEE. 2022, pp. 1–2.

[4] Soyed Tuhin Ahmed and Mehdi B. Tahoori. "Compact Functional Test Generation for Memristive Deep Learning Implementations using Approximate Gradient Ranking". In: *2022 IEEE International Test Conference (ITC)*. 2022, pp. 239–248. DOI: 10.1109/ITC50671.2022.00032.

[5] Soyed Tuhin Ahmed, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Anghel Lorena and Mehdi B Tahoori. "Binary bayesian neural networks for efficient uncertainty estimation leveraging inherent stochasticity of spintronic devices". In: *NANOARCH'22: 17th ACM International Symposium on Nanoscale Architectures*. ACM. 2022, pp. 1–6.

[6] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Scalable Spintronics-based Bayesian Neural Network for Uncertainty Estimation". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2023, pp. 1–6.

[7] Soyed Tuhin Ahmed, Roman Rakhmatullin and Mehdi B Tahoori. "Online Fault-Tolerance for Memristive Neuromorphic Fabric Based on Local Approximation". In: *2023 IEEE European Test Symposium (ETS)*. IEEE. 2023, pp. 1–4.

[8] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "Testing Spintronics Implemented Monte Carlo Dropout-Based Bayesian Neural Networks". In: *2022 IEEE European Test Symposium (ETS)*. IEEE. 2024, pp. 1–6.

[9] Soyed Tuhin Ahmed, Kamal Danouchi, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "NeuSpin: Design of a Reliable Edge Neuromorphic System Based on Spintronics for Green AI". In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2024, pp. 1–6.

[10] Soyed Tuhin Ahmed*, Surendra Hemaram* and Mehdi B Tahoori. "NN-ECC: Embedding Error Correction Codes in Neural Network Weight Memories using Multi-task Learning". In: *2024 IEEE 42th VLSI Test Symposium (VTS)*. IEEE. 2024, pp. 1–6.

[11] Soyed Tuhin Ahmed, Kamal Danouchi, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Enhancing Reliability of Neural Networks at the Edge: Inverted Normalization with Stochastic Affine Transformations". In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2024, pp. 1–6.

[12] Soyed Tuhin Ahmed, Michael Hefenbrock and Mehdi B. Tahoori. "Tiny Deep Ensemble: Uncertainty Estimation in Edge AI Accelerators via Ensembling Normalization Layers with Shared Weights". In: *2024 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2024, pp. 1–9.

[13] Soyed Tuhin Ahmed and Mehdi Tahoori. "Few-Shot Testing: Estimating Uncertainty of Memristive Deep Neural Networks Using One Bayesian Test Vector". In: (2024). arXiv: 2405.18894 [cs.LG].

First-author journal papers that are part of this thesis

[14] Soyed Tuhin Ahmed, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "Neuroscrub+: Mitigating retention faults using flexible approximate scrubbing in neuromorphic fabric based on resistive memories". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.5 (2022), pp. 1490–1503.

[15] Soyed Tuhin Ahmed, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "Design-time Reference Current Generation for Robust Spintronic-based Neuromorphic Architecture". In: *ACM Journal on Emerging Technologies in Computing Systems* 20.1 (2023).

[16] Soyed Tuhin Ahmed, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "SpinDrop: Dropout-Based Bayesian Binary Neural Networks With Spintronic Implementation". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 13 (2023). DOI: 10.1109/JETCAS.2023.3242146.

[17] Soyed Tuhin Ahmed and Mehdi B Tahoori. "Fault-tolerant Neuromorphic Computing with Memristors Using Functional ATPG for Efficient Re-calibration". In: *IEEE Design & Test* (2023).

[18] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures". In: *ACM Transactions on Embedded Computing Systems* 22.5s (Sept. 2023), 131:1–131:25. ISSN: 1539-9087. DOI: 10.1145/3609116. URL: https://doi.org/10.1145/3609116.

[19] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Spatial-SpinDrop: Spatial Dropout-based Binary Bayesian Neural Network with Spintronics Implementation". In: *IEEE Transactions on Nanotechnology* (2024).

[20] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Scale-Dropout: Estimating Uncertainty in Deep Neural Networks Using Stochastic Scale". In: *arXiv preprint arXiv:2311.15816* (2024).

[21] Soyed Tuhin Ahmed and Mehdi B Tahoori. "One-shot online testing of deep neural networks based on distribution shift detection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[22] Soyed Tuhin Ahmed and Mehdi B Tahoori. "Concurrent Self-testing and Uncertainty Estimation of Neural Networks Using Uncertainty Fingerprint". In: *arXiv preprint arXiv:2401.01458* (2024).

Co-author papers that are not part of this thesis

The following list enumerates papers by the author of this dissertation that are not part of this thesis.

[23] Surendra Hemaram, Soyed Tuhin Ahmed, Mahta Mayahinia, Christopher Münch and Mehdi B Tahoori. "A Low Overhead Checksum Technique for Error Correction in Memristive Crossbar for Deep Learning Applications". In: *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE. 2023, pp. 1–7.

[24] Atousa Jafari, Mahta Mayahinia, Soyed Tuhin Ahmed, Christopher Münch and Mehdi B Tahoori. "MVSTT: A Multi-Value Computation-in-Memory based on Spin-Transfer Torque Memories". In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE. 2022, pp. 332–339.

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben haben und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, 1. June 2024

Soyed Tuhin Ahmed

Contents

1. Introduction

In recent years, neural networks (NNs) have revolutionized the field of artificial intelligence (AI), driving significant advancements in various domains such as image recognition, natural language processing, and autonomous systems [1]. Their ability to learn from vast amounts of data and perform complex tasks with high accuracy has led to widespread adoption in industry and academia. Applications of NNs span from consumer electronics [2, 3] to healthcare [4, 5], where they assist in diagnostic procedures, and to autonomous vehicles, where they enable safe navigation. It is forecasted that by 2030, AI could potentially generate an additional $13 trillion in global economic activity, or approximately 16% higher cumulative GDP in comparison to 2018. Furthermore, 70% of companies are expected to adopt at least one type of AI technology [6].

Despite their recent popularity, the history of neural networks dates back to the 1940s, when neurophysiologist Warren McCulloch and mathematician Walter Pitts modeled a simple neural network using electrical circuits to describe how neurons in the brain might work [7]. The recent advancement of NN can be attributed to the availability of large-scale data [8], the improvement of NN algorithms that allow the size (depth and width) of models to increase [9, 10, 11], the availability of cloud computing platforms [12], and performing computation in parallel processing architectures such as graphics processing units (GPUs) [13].

However, to deploy an NN for inference, it is first trained on a dataset for a specific task to obtain trained parameters that are stored in conventional or emerging memory technologies for inference. Then, in the inference phase, the trained model is put into action on novel data (expected to be from the same distribution as training data) to make actionable predictions. The overall training-to-deployment flow is depicted in Fig. 1.1.

Therefore, the large size of NN leads to a) a large number of model parameters that need to be stored in memory, which can be 138 million for the VGG-16 topology [12] and b) substantial multiply and accumulate (MAC) operations, the fundamental computation of NN, which can be 5112.78 million for the VGG-16 topology [14]. Therefore, the inference of an NN based on its inputs can be heavily demanding in terms of memory, energy, and computation capability [1, 15].

Therefore, cloud servers, which provide very powerful computational and large storage capabilities, are a natural fit for the inference stage of NNs. However, the cloud-based ecosystem has many limitations that prevent the adoption of all AI applications. Specifically, for edge AI devices (see Fig. 1.2), sending data to the cloud for inference and receiving the results is not practical due to prediction latency, bandwidth constraints, and privacy concerns. Many delay-sensitive AI applications, such as autonomous driving, industrial control systems, and robotics, require fast processing of the incoming data to produce real-time results. Consequently, extremely high network bandwidth will be necessary for cloud-based processing

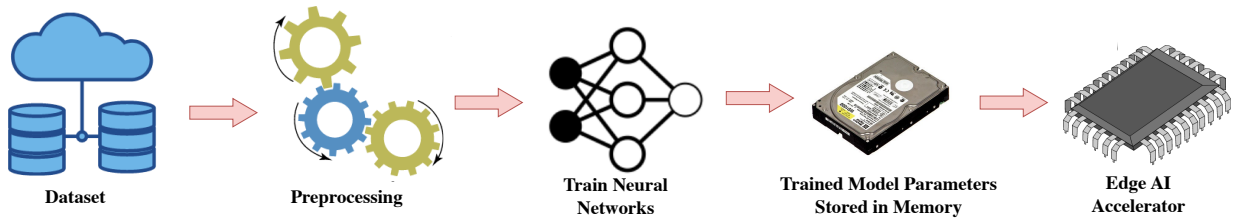


Figure 1.1.: The training flow of NNs.

Figure 1.2.: A few examples of edge AI devices and applications, including safety-critical applications such as autonomous driving and industrial predictive maintenance.

for edge AI, which is challenging due to varying network qualities (e.g., in remote locations). Therefore, computation near or on the edge device is preferred.

1.1. Edge AI

Edge AI refers to the deployment of AI algorithms fully or partially on edge devices, such as smartphones, IoT devices, and autonomous vehicles, rather than relying on centralized cloud servers for inference. This paradigm shift aims to address the limitations of cloud-based AI by bringing computation closer to the data source. Edge AI can be categorized into several types based on how computation tasks are partitioned: server-based edge inference, device-edge joint inference, and fully on-device edge inference.

In server-based edge inference, the entire AI models are deployed on edge servers. For inference, edge devices upload their input data to edge servers and receive inference results. However, data communication latency and user data privacy are still a concern. Specifically, input data can be personal information or the record of street-view video. Therefore, sending them to the cloud raises privacy issues.

On the other hand, in device-edge joint inference, AI models are partitioned into edge devices and servers. Specifically, the first few layers are deployed locally on the edge device, producing a local output with simpler processing, and transmitting the local output to an edge server. Afterward, the final inference results are transferred back to the edge devices. Thus, user data privacy can be preserved, but latency is still a concern.

Consequently, the most attractive solution is fully on-device inference, where computations are done locally on the device without any data transmission to external servers. Therefore, concerns about user data privacy and latency are mitigated. As a result, this thesis focuses on fully on-device inference. However, edge devices are resource-constrained and battery-powered, making a fully on-device inference particularly challenging.

Therefore, more specialized hardware for accelerating NN is necessary to overcome the challenges of computation and power efficiency.

1.2. Edge AI Accelerator

Several AI accelerator architectures have been developed, from more generic hardware such as Graphic Processing Units (GPUs) to highly specialized hardware such as Tensor Processing Units (TPUs) [16], and Application Specific Integrated Circuits (ASICs). Those solutions are largely focused on accelerating the MAC operations of NN models. However, since NNs are data-centric applications, the data movement between the processing unit and the memory unit becomes the bottleneck of the entire task. This leads to the well-known memory wall problem [17].

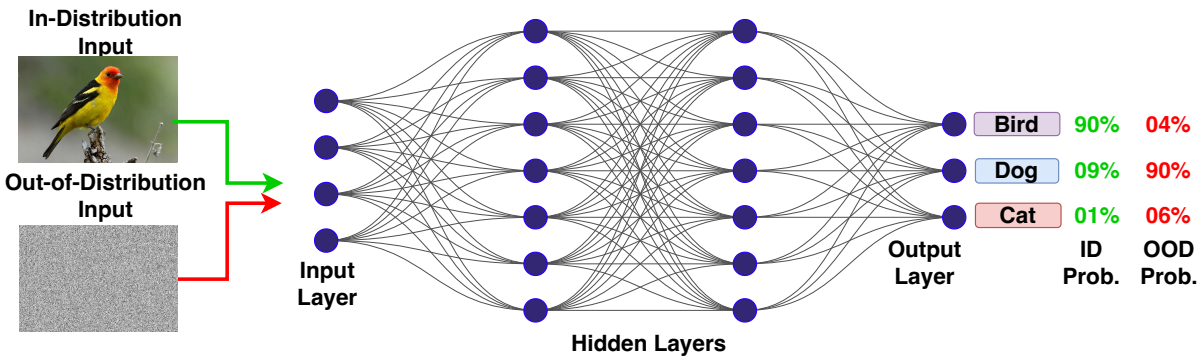


Figure 1.3.: Visualization of the uncertainty of 3-class (Cat, Dog, and Bird) NN when it receives in-distribution (ID) input of bird and out-of-distribution (pure Gaussian) input. The model predicts Gaussian noise as a Dog with high confidence.

In order to overcome this problem, conventional parallel computing architectures typically employ multi-threading to hide memory access latency [18]. However, latency cannot be easily hidden in AI computation, and intelligent memory architecture needs to be considered.

Consequently, *Computation in-Memory* (CiM) architectures with emerging resistive non-volatile memory technologies (referred to as memristors) promise to mitigate the data transfer overhead by performing MAC where data already reside. Consequently, the inference speed is improved, and the energy consumption is reduced. Furthermore, memristors offer many benefits, such as highly scalable designs, non-volatility, and low power consumption, compared to CMOS-based conventional static and dynamic memories. Furthermore, some memristor technology offers high writing speed (within a few nanoseconds), exceptional endurance, and demonstrates compatibility with established semiconductor manufacturing processes [19].

1.3. Uncertainty In AI Accelerators

Despite the computational and energy efficiency benefits that edge AI accelerators offer, reliability is still a concern as they are often deployed in dynamic environments with varying conditions. Reliability is especially critical for safety-critical applications such as autonomous vehicles, industrial control, aviation, medical imaging, and surgical robots, where malfunctions could directly lead to severe consequences or even loss of life. However, these systems are inherently susceptible to uncertainties stemming from various sources, including inference data, model architecture, hardware architectures, memory technology, NN computation, and external environmental factors.

1.3.1. Uncertainty In Predictions

During inference, the edge AI accelerator is not guaranteed to receive input from the same distribution as the training data. For example, an NN model trained on MNIST (handwritten digits from zero to nine) can receive input that does not resemble a handwritten digit. Such data are referred to as out-of-distribution (OOD) data samples. OOD data can be due to: 1. Data distribution shifts, where the statistical properties of the input data change due to factors such as sensor noise or environmental variations [20], 2. Domain shifts, where changes in the operational environment cause the data to differ significantly from the training domain, 3. Open-world classes, where new classes of data emerge during inference that were not defined during training.

In the presence of OOD data, NNs can make inaccurate, unpredictable, and uncertain predictions with high confidence, as shown in Fig. 1.3. Existing work has shown that when an MNIST-trained NN model receives OOD input (pure Gaussian noise), the mean prediction probability of 10000 inputs is 91% [21],

indicating the model's overconfidence despite not recognizing the input as belonging to any of the trained classes. This overconfidence can be catastrophic for safety-critical AI applications. In fact, the first fatal accident involving an autonomous vehicle that killed a Tesla driver was linked to the system's inability to correctly identify an OOD object (a white truck against a bright sky), highlighting the critical need for OOD detection and mitigation in safety-critical edge AI [22].

Therefore, in safety-critical AI applications, providing predictive uncertainty estimates in addition to model perdition is paramount to increasing confidence in prediction and improving overall reliability.

1.3.2. Functional Uncertainty In Edge AI Accelerators

Since edge AI accelerators are deployed in real-world environments, they are susceptible to single and multiple permanent and soft faults and online variations that lead to errors [23] in weights, MAC results, and activations. Specifically, online faults can be due to fluctuations in operating temperature, radiation level, electromagnetic interference (EMI), and voltage and current fluctuations [24, 25, 26]. In addition, some faults and defects occur during manufacturing, but they escape manufacturing testing, leading to online faults. On the other hand, the conductance of emerging memory cells follows a distribution rather than a single point value due to manufacturing variations, and it further fluctuates due to operating temperature [27]. Consequently, noise is introduced in the MAC results.

These faults can occur in all layers or in any intermediate layer. However, their effect would ultimately cause a mismatch in the output distribution from a fault-free one. This is because faults in any intermediate layers are likely to flow to the output.

However, the impact of faults and variations usually depends on the amount, location, and type of faults. Specifically, if the perturbations are minor, NN can tolerate them [28]. However, their accuracy decreases significantly with a large-magnitude hardware error [29, 30] or when faults occur in more sensitive locations, making edge AI accelerators non-functional and their prediction uncertain.

Consequently, the introduction of these non-idealities adds another layer of uncertainty to the system. Specifically, uncertainties are introduced into the functionality of edge AI accelerators, e.g., fault-free weight storage, MAC results computation, activations computation, and prediction computation. Therefore, at a given time after post-mapping, it is uncertain whether the edge AI accelerators are functionally correct. Although challenging, this form of uncertainty is important to quantify (explicitly or concurrently) using a proper online testing methodology and mitigate using efficient reliability improvement methods, especially in safety-critical applications.

1.3.3. Uncertainty Due to Aging

Edge AI accelerators, like any electronic system, are subject to time-dependent uncertainty stemming from aging and wear-out. Specifically, data retention and endurance of memory cells are significant concerns. In data retention faults, the stored data is not retained after a certain amount of time due to external influence [29]. Also, in the multi-level memristors, the conductance value drifts due to external influence. These faults accumulate over time after the initial write operation (weight mapping) and significantly influence the performance of edge AI accelerators [29]. Specifically, the accuracy of an edge AI accelerator drops drastically over its expected operating time or the expected time before the next weight matrix update if retention faults are not mitigated [31].

On the other hand, certain memristor technologies, e.g., resistive random access memory (RRAM), have limited endurance, typically ranging between 10^5 and 10^7 [32]. When the endurance limit is reached, the memory cells may not be programmable [33], manifesting as permanent faults. Assuming that with age the number of synaptic weight updates increases, the endurance of the memristor also introduces uncertainty and is a reliability concern.

1.4. Reliability Requirements for Safety-Critical Applications

Several reliability requirements must be met, especially in safety-critical applications, in order to deploy AI applications to edge AI accelerators that are susceptible to various uncertainties. These applications require robust and consistent performance under unpredictable and challenging conditions. Consequently, edge AI accelerators should maintain acceptable performance in the presence of faults, either through error detection and correction mechanisms or by self-healing mechanisms that allow graceful degradation in inference accuracy. Moreover, ensuring consistent performance despite fluctuations in external environmental factors, e.g., operating temperature and device aging, is paramount for functional safety and overall fault tolerance. In addition, monitoring the fault status of AI accelerators through proper online testing and adapting as needed to ensure reliability is important.

Furthermore, providing reliable uncertainty estimates related to each prediction allows informed decision-making and evaluation of potential risks. Also, identifying OOD data is important, as it allows experts to review these predictions before reaching end users.

Lastly, the ability to quickly and accurately identify the root causes of uncertainty allows for the facilitation of proper mitigation strategies. For example, identifying sources of uncertainty can aid in targeted uncertainty mitigation strategies, such as hardware maintenance in terms of fault removal, re-calibration, or re-training.

Therefore, key reliability considerations that are relevant to this thesis include fault tolerance, predictive uncertainty estimates, OOD data detection, robustness to environmental variations, root cause analysis, and testability, which is periodic and concurrent monitoring of the fault status of AI accelerators.

1.5. Challenges and Research Direction

To ensure the reliable online operation of AI accelerators, lightweight yet effective uncertainty estimation and reduction approaches suitable for implementation on resource-constrained and battery-powered systems are required. However, several challenges are involved in uncertainty estimation, uncertainty reduction, and testing of edge AI accelerators. In particular, conventional approaches are not resource-scalable, and we elaborate on specific challenges.

1.5.1. Predictive Uncertainty Estimation

For uncertainty estimation, model ensembles and Bayesian neural networks (BayNNs) are inherently suitable. However, these methods are more resource-intensive than conventional NNs, making them challenging to implement in resource-constrained edge AI accelerators.

Specifically, the model ensemble method, although considered the gold standard for uncertainty estimation, requires training, storing, and processing multiple models. Consequently, for M ensemble members, the number of MAC operations, memory, latency, and power consumption is $M\times$ compared to a single model. If there are M AI accelerators available, then the inference latency can be reduced to the same as a single model, but the MAC operation and power consumption still remain an issue.

On the other hand, obtaining the posterior distribution of BayNN is intractable analytically as it requires integrating over the entire parameter space. Thus, approximation methods such as Variational Inference (VI) [34] and Monte Carlo (MC) Dropout [35] are widely used. Nevertheless, computational resource and memory requirements are still an issue. Implementing BayNN algorithms in CiM architectures can mitigate some of their inherent costs, and the benefits of bayNN, in-memory computation, and memristor can be obtained in a single package. Therefore, the target of this thesis is to implement BayNNs in the CiM architecture. However, it presents several challenges.

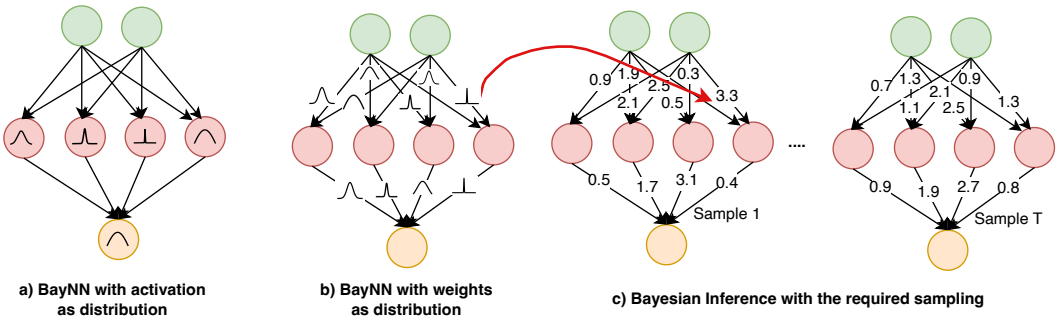


Figure 1.4.: BayNN with variational distribution in a) activation and b) weights. During each forward pass, c) element-wise sampling results in a single-point value for weights/activation, on which computations are performed for a forward pass.

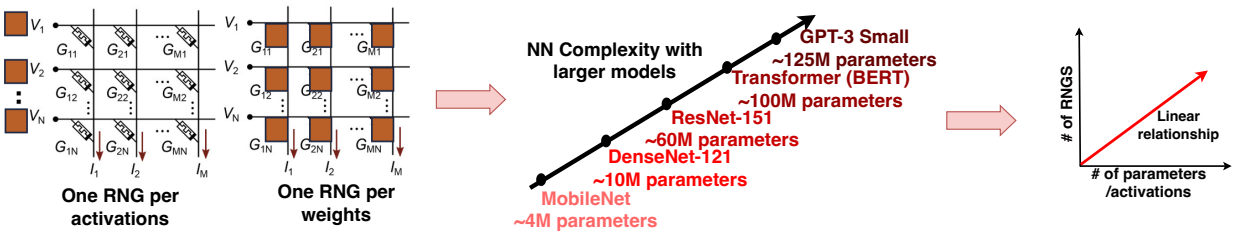


Figure 1.5.: The resource scalability issue of BayNN implementations in CiM architectures. The number of RNGs increases with the size of the model, which can lead to millions of RNGs in a larger model. **It is recommended to view this figure in color.**

In VI, weights or neurons are represented as distributions, as shown in Figs. 1.4 (a) and (b). During each forward pass, sampling is required from these distributions, as shown in Fig. 1.4 (c). Subsequently, the MAC operation or activation calculation is performed using the sampled value. Implementing weights or neuron distributions in CiM and efficiently sampling from them for the inference step is a challenge. In a conventional VI-based BayNN implementation in the CiM architecture, a random number generator (RNG) is required for each weight or activation representation, as shown in Fig. 1.5. Consequently, the number of RNGs required in an edge AI accelerator can be millions. Since each RNG requires a certain chip area and has power consumption, overall overhead can be prohibitive. In other implementation methods, memory consumption is an issue as parameters of the distribution, e.g., mean and variance, require storage.

On the other hand, MC-Dropout is particularly attractive because it has the same memory consumption and single-value parameter representation as a conventional NN. However, it requires applying Dropout to each neuron [35] or weights [36]. Consequently, the number of Dropout modules required in an edge AI accelerator can be millions as the number of Dropout modules equals the total number of weights or activations in the model. Similar to VI, it leads to huge overhead as each Dropout module requires a certain chip area and has power consumption.

Consequently, implementing uncertainty estimation methods in edge AI accelerators leads to resource scalability issues. Since the resources required to implement BayNNs grow significantly with the size of the model, the scalability of large models in CiM architectures is hindered. This means that achieving the desired performance with minimal energy consumption, memory usage, and latency at the same time is a challenge for BayNNs.

1.5.2. Quantification of Functional Uncertainty of The Edge AI Accelerators

Similarly, testing edge AI accelerators presents several challenges. Since NN can have over 100 million parameters, structural testing such as the March test, which requires reading and writing each individual cell, is difficult or costly. Therefore, in this thesis, we focus on functional testing, which ensures that the functionality of the model is as expected. However, there are still some challenges.

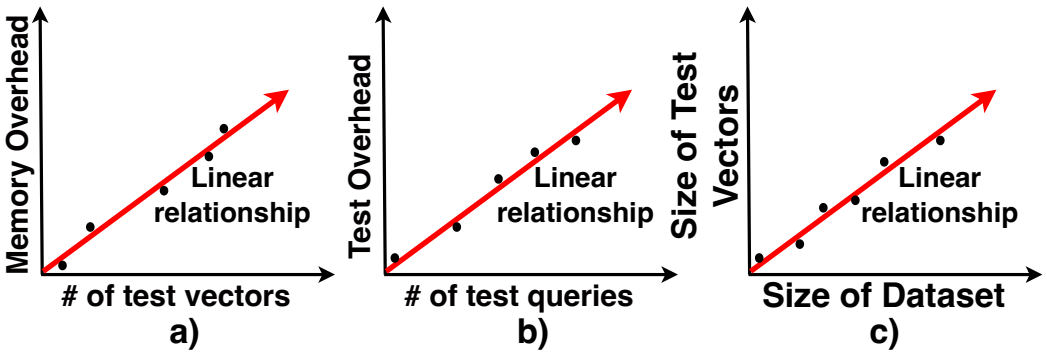


Figure 1.6.: Resource scalability issue of testing edge AI accelerators in conventional methods, where a) storage overhead increases linearly with the number of test vectors, b) testing overhead in terms of latency, the number of MAC operations, and power consumption increases linearly with the number of the test queries (forward pass), and c) size of test vectors increases linearly with the size dataset.

Specifically, test vector compaction and a small number of test queries (forward passes) are preferred in explicit testing scenarios. This is because the test overhead in terms of storage increases linearly with a number of test vectors, as shown conceptually in Fig. 1.6 (a). On the other hand, the number of MACs, power consumption, and latency increase linearly with the number of test queries, as shown conceptually in Fig. 1.6 (b). Furthermore, testing latency is critical since the system is non-functional while testing. This is because real-time and always-on AI applications cannot tolerate long system downtime, and a shorter testing latency is essential. However, in existing functional testing methods [37], all of the data points from the validation dataset are utilized for testing the edge AI accelerators, which, even for a smaller CIFAR-10 dataset, involves 10000 images with a shape 32×32. Consequently, the testing overhead increases linearly with the size of the dataset, as shown conceptually in Fig. 1.6 (c). Therefore, this again leads to the resource scalability issue for larger NN datasets.

Furthermore, in terms of test vector generation, treating the entire NN as a black box without introducing any back door to the model, fine-tuning the model, giving access to parameters or intermediate results, and testing a pre-trained model is preferred. This is because many users consider pre-trained NNs to be intellectual property (IP), particularly with pre-trained models from Machine Learning as a Service (MLaaS). The global MLaaS market is expected to surpass 200 billion USD by 2028 [38]. In those scenarios, introducing a back door to the model or fine-tuning the model, IP rights may be violated. On the other hand, giving access to parameters or intermediate results may make the model susceptible to model extraction attacks or fault attacks and can lead to potential IP theft.

On the other hand, the training process for complex NN models can be cumbersome and expensive. However, many users do not have access to computational resources or larger datasets for such a complex NN model. Thus, many industry-leading companies, such as Microsoft Azure and Amazon AWS, and start-up companies, such as pretrained.ai, are offering pre-trained models for their users. Pre-trained models allow the user to map the models to AI accelerators and make predictions on inference data right away. Therefore, testing such a model is desirable. Consequently, no knowledge or modification of the training process is desired, and the baseline accuracy of each model under test should remain unchanged for test generation.

In addition, many of the existing testing methods propose to introduce additional hardware, such as runtime monitoring of currents [39]. However, modifying the hardware architecture for testing is not preferred as it allows testing off-the-shelf edge AI accelerators. In addition, additional hardware may introduce additional hardware overhead.

Therefore, for explicit testing, the goal of this thesis is to online test the entire NN model implemented in the edge AI accelerator by a) compacting test patterns, b) treating the NN as a "black box" with only model

outputs (or logits) requiring access, c) testing pre-trained models without changing any parameters or activations, and d) not making any changes to the existing edge AI accelerator architecture.

On the other hand, for concurrent testing, traditional concurrent error detection (CED) methods, such as error-correcting codes (ECC), are not suitable for detecting multiple, permanent, and logic faults, especially in buffer memory storing activations. Additionally, testing methods that do not require fault injection studies and have low false positive rates are preferred. This is because fault injection studies can be computationally expensive, are targeted for specific scenarios, and may not accurately represent other real-world fault scenarios. Also, high false positive rates can raise too many false alarms. Consequently, for concurrent testing, the aim of this thesis is to test for multiple, permanent, memory, and logic faults in edge AI accelerators with low false rates and high coverage.

Existing Spintronics-based BayNN methods focus on the implementation of BayNN, reducing power consumption, circuit, and system design. However, proper testing to ensure the functional correctness of BayNN computation and Dropout generation is ignored. Consequently, this thesis also aims to test Spintronics-based BayNN.

In general, the proposed approaches act as the first line of defense and complement more sophisticated testing that has a larger overhead, e.g., to find fault locations in a crossbar or a faulty unit. Therefore, more sophisticated testing can be performed on demand. Specifically, low-cost testing is performed more frequently, and sophisticated testing is performed on demand. Consequently, our approach can lead to a form of "hybrid testing" based on testing overhead. Note that hybrid methods are used extensively in industry to save energy, improve computational efficiency, and so on. For example, to optimize power efficiency, modern computer architectures such as ARM big-little utilize high-performance cores (big) with energy-efficient cores (little) in a single processor [40]. Extending this concept to the testing of AI accelerators can optimize testing latency, memory overhead, and power consumption.

1.5.3. Uncertainty Reduction

For uncertainty reduction, methods such as ECC and redundancy are conventionally used. However, they are challenging to implement in resource-constrained edge AI accelerators. For example, ECC typically incurs a 7-15% overhead for parity bit storage, depending on the specific ECC scheme and data word size. However, since NNs can have millions of parameters, this overhead is impractical for resource-constrained edge AI applications. Similarly, conventional redundancy methods such as triple modular redundancy, which involve triplicating hardware or computations to ensure reliability, can result in a threefold increase in memory and computational requirements. They are also impractical for resource-constrained edge AI accelerators. Therefore, low-cost reliability solutions must be considered for edge AI systems.

Furthermore, a memristor-based CiM architecture can introduce noise to the MAC results of a layer and consequently change the distribution of a neuron from its trained one due to manufacturing and infield variations. Reduction of such uncertainty is imperative for reliable performance.

In addition, the reliable operation of Bayesian NNs in CiM architectures is crucial. However, the primary focus of existing CiM-implemented Bayesian NNs has largely been on uncertainty estimation rather than improving *inherent fault tolerance* against non-idealities in CiM architectures. On the contrary, existing fault tolerance studies that use the Bayesian approach have a) overlooked the uncertainty estimation aspect, b) are targeted for a specific non-ideality type, and c) require computationally expensive neural architecture search (NAS). Therefore, designing a BayNN that is self-immune to non-idealities of CiM architecture without sacrificing the uncertainty estimation aspects is a challenge.

1.5.4. Integration Challenges

NN model is often developed and trained with full precision 32-bit. However, memristor devices have limited stable states. Therefore, quantization-aware training or post-training quantization must be adopted. However, the performance of NN can be reduced due to quantization error, especially in binary NN. Therefore, keeping the performance close to the baseline is challenging. Furthermore, certain computations, such as scaling of weights to reduce quantization error, are not feasible in certain edge AI accelerators such as CiM. Therefore, training must be further adopted to overcome the constraint of edge AI accelerators while still maintaining high performance.

1.6. Summary of Contributions

The objective of this thesis is to overcome the challenges mentioned using scalable and low-cost yet effective methods. Specifically, we explore algorithm-hardware co-design-based solutions to improve the uncertainty quantification, testability, reliability, performance, manufacturing yield, and efficiency of edge AI accelerators.

The contributions of this thesis are categorized into methods for a) uncertainty estimation, b) testing, c) uncertainty reduction, and d) ensuring continuous availability. The summary of contributions of this dissertation are as follows:

Predictive Uncertainty Estimation

- **Binary Bayesian Neural Networks**: To overcome the limited stable states of Spintronics memory, we proposed MC-Dropout on binary neural networks. Binary neural networks (BNN) use only two states to represent weights and activation. Thus, proposed binary Bayesian neural networks (BinBayNN) can be directly implemented in existing CiM architecture with any modifications to their crossbar structure. However, direct implementation of MC-Dropout on BNN is not feasible. Therefore, we propose a learning objective function that is mathematically equivalent to full precision MC-Dropout.
- **Spintronic Dropout Model**: Due to the analog nature of the Spintronics-based Dropout module, variation also affects the dropout rate. We propose to model and train the MC-Dropout method with the dropout rate as a distribution rather than a single point value to account for variations. Evaluation of our approach shows comparable performance with slight improvement in accuracy in some cases.
- **Reduce the number of Dropout modules**: To reduce the number of Dropout modules and scale our BinBayNNs to convolutional neural networks (CNNs), we first proposed to drop group neurons concurrently. Dropping group neurons concurrently also simplifies circuit design for the Dropout module, as knowledge about the spatial location of the kernels is not required. Furthermore, to reduce the number of Dropout modules to **one for the entire NN model**, regardless of the size of the model, in the following work, we proposed to drop another parameter group of BNN. Specifically, instead of Dropping neurons or weights, we propose to drop the scale vector. However, instead of dropping the scale vector element-wise or group-wise, we propose the vector dropout concept, where the entire vector is dropped concurrently with a probability. To employ each of the Dropout strategy, we also propose CiM architecture and parameter mapping strategy.
- **Novel Dropout approach**: We propose the concept of unitary Dropout. In a unitary Dropout, instead of dropping an element to zero as done in a conventional dropout, we propose dropping it to one. Consequently, our approach allows for the simultaneous dropping of all the elements of a vector but still allows information flow through the NN. Otherwise, total loss of information occurs as the activation of a layer becomes zero.

- **In-Memory centric Bayesian approximation**: To efficiently map the distributions of BayNNs with VI approximation to CiM architecture, we proposed the Bayesian in-memory approximation method. Our approach provides a memory-friendly distribution that can be efficiently mapped and sampled in the CiM architectures. We empirically demonstrate that our approach performs similarly to the original distribution of variational inference in terms of predictive performance and uncertainty estimates.
- **Novel Bayesian Neural Networks Topology**: We propose a novel BayNN topology to allow efficient sampling from CiM architectures. Our proposed topology is scalable to any existing NN topology and requires only **three RNGs for the entire model**, regardless of the size of the model. Furthermore, our approach requires only minor changes to the peripheral circuits of CiM architecture and during activation computation. Thus, existing CiM architectures and NN topologies can be employed.
- **Bayesian Subset Parameter Inference**: We introduce the Bayesian subset parameter inference method, where both deterministic and stochastic treatments are applied to the parameter groups of NN. Specifically, we propose treating the smallest parameter group of NN as Bayesian while treating the rest of the parameter group as deterministic. Therefore, our approach allows the first instance of BinBayNNs with VI, significantly reducing the memory overhead and allowing weight mapping to existing spintronics-based memory arrays in the CiM architecture. In a naive approach, discretizing the distributions to binary values will significantly reduce accuracy. We also proposed a CiM architecture for our approach with two crossbars. In one crossbar, the MAC operation is performed, and in another crossbar, a scale vector is stored. MAC results scaling is performed stochastically in the peripheral circuits.
- **Tiny-Deep Ensemble**: We propose the tiny deep ensemble network for ensemble-based uncertainty estimation in resource-constrained edge AI accelerators. In our approach, we only propose to ensemble the normalization layers that consume $1-2\%$ of the overall parameters. The rest of the parameters are shared among the ensemble members, leading to approximately the same memory overhead as a single model. Furthermore, we also propose a method that allows parallelizing the training and inference to reduce the latency to the same as a single model. In addition, we propose a CiM architecture for the implementation of our proposed approach.

Uncertainty Estimation of The Functionality Edge AI Accelerators

- **Low-overhead Online Functional Testing**: We proposed several test pattern compaction methods and online testing methods to reduce the overall cost of online functional testing edge AI accelerators. Here, 'online' refers to post-mapping but pre-deployment testing as well as post-deployment testing, e.g., infield testing.
  - **Approximate Gradient Ranking Method**: We introduce the concept of the approximate gradient ranking method. Our approach ranks training data points based on their approximated overall accumulated gradient values for the entire training. Our approach does not make any changes to the conventional training curve or AI accelerator architecture, is non-invasive, and is a black-box testing method that can also test pre-trained models. Thus, baseline accuracy is preserved, and a high test coverage can be achieved using only 15 to 63 test queries. We also propose test application methods based on the severity of faults and can detect hard-to-detect faults.
  - **Single-Shot testing**: We introduce the single-shot testing method to further reduce test overhead. Our approach requires a single test vector and forward pass on the edge AI accelerator to test the entire model. We propose a learning algorithm that can generate the proposed single-shot test vectors and an online test application method. Our approach has the added benefit of not requiring access to the training data, as the training data is not always available during inference, especially for a pre-trained model.

  - **Few-Shot Testing with Single Bayesian Test Vector**: We proposed the concept of few-shot testing that can test a large model, i.e., a model with a large number of classes, in a single shot but can also test a small model with a few shots. In our approach, we introduce the concept of the Bayesian test vector, where each element, e.g., pixels for an image, is a distribution rather than a single point value. Therefore, infinite samples can be taken from a single Bayesian test vector. We also proposed a learning algorithm to generate the proposed Bayesian test vector and the online test application method.

- **Testing Dropout-based BayNN Implemented in Spintronics-based CiM**: We introduced the repeatability ranking based automatic-test-pattern-generation (ATPG) method and online test application method to test Spintronics implemented Dropout-based BayNNs. Our approach overcomes the challenge of testing BayNN using only 100 test vectors with consistently high test coverage for critical faults and low false-positive rates.

- **Concurrent Testing of NNs**: We propose an NN topology with dual heads where one head produces model prediction, and another head gives the runtime fingerprint of fault status. The model can be classified as faulty or fault-free by comparing the runtime fingerprint with the fault-free fingerprint. A learning algorithm with an objective function is proposed for the proposed concurrent testing that does not impact baseline accuracy.

- **Disentangling Source of Uncertainty**: We introduce a method for disentangling the source of uncertainties during online operation. Our method effectively identifies whether the predictive uncertainty is due to out-of-distribution data or hardware faults, which is overlooked in existing works. It uses predefined disentanglement test vectors and compares the resulting model fingerprints with fault-free fingerprints. The evaluation of the proposed method shows high accuracy in distinguishing between different sources of uncertainty.

Uncertainty Reduction

- **Self-Healing Approaches:**

  - **Self-Healing Bayesian Neural Networks**: We propose a novel normalization and Dropout layer for *inherently self-healing* BayNN that a) does not require any implicit non-idealities modeling, b) is generalizable across different memristor technologies and their non-idealities, c) is easy to train, d) CiM implementation friendly, and e) is still able to provide uncertainty estimates without reducing its accuracy. Our approach, instead of dropping individual neurons, introduces implicit additive and multiplicative noise into the MAC results of each layer during training for robustness during inference. Evaluation of various AI tasks, NN classes, and benchmark datasets show significant improvement in accuracy.

  - **Self-Healing the Impact of Manufacturing and Infield Thermal Variations**: We propose to binarize each partial sum of a layer to increase the sensing margin. Consequently, it allows self-healing from the impact of manufacturing variations. Furthermore, we proposed a design-time reference current generation algorithm that allows the NN to self-heal from the impact of in-field thermal variation for the entire operating temperature range (up to 125 °C).

- **Periodic Maintenance Approaches**:

  - **Runtime Re-Calibration For Fault-Tolerance**: We proposed a low-cost re-calibration approach that re-calibrates the batch normalization layer in the presence of non-idealities and in a per-chip manner. The approach is grounded on the fact that the non-idealities of memristors shift activation distributions of a neuron from training distributions, affecting post-mapping inference accuracy. Therefore, to reduce the overall re-calibration overhead that is not addressed in existing works, we propose the approximate batch normalization method that completely removes the batch-normalization calculation during inference and requires much simpler calculations for the in-field re-calibration step. Furthermore, we also

proposed an automatic test pattern generation method for compacting the re-calibration inputs. Consequently, our approach requires only 0.2% of training data for re-calibration. Furthermore, we proposed a "partial re-calibration" method that can reduce the overall overhead by re-calibration only a part of the network. Therefore, our low-latency re-calibration allows the NN to resume normal operation faster.

  - **Maintaining Retention Faults and Aging Induce Drifts**: We introduced the concept of approximate scrubbing in memristor-based CiM architecture. In our approach, a retention-aware weight mapping is proposed, where stable weights are mapped to a predefined area called the scrub region, and unstable weights are mapped to another region, the non-scrub region. The scrub region is periodically scrubbed to restore the respective values. The scrubbing rate is proposed to be adjusted based on the probability of retention faults and memristor technology. We also proposed a retention-aware training algorithm and objective function that encourages the weights to be grouped based on the scrubbing preference. Consequently, our approach can maintain the baseline inference accuracy with virtually zero storage overhead.

- **Guaranteed Soft-Faults Correction for Digital AI Accelerators**: We propose a zero overhead ECC scheme for guaranteed soft-fault correction in digital AI accelerators. Specifically, we proposed an efficient and generalized method that embeds the ECC parity bits in the NN weight matrix during NN training, using a multi-task learning method that keeps the same inference accuracy as the baseline. Our approach completely eliminates the parity bit storage overhead for different ECC schemes. Therefore, we provide multi-bit error correction guarantees, which is imperative for safety-critical AI applications. The evaluation of the proposed method shows significantly improved fault tolerance with single- and multi-bit ECC on different NN models and benchmark datasets.

Ensuring continuous availability Sophisticated testing and reliability improvement methods, such as re-training, can be time-consuming. However, the NN is non-functional when such maintenance is done, and always-on applications do not tolerate such system downtime. Therefore, we propose an online fault tolerance method to ensure the availability of neural networks that are implemented in the memristive CiM architecture. Instead of creating copies of the entire network, we propose protecting only the fault-sensitive part of the NNs and applying compression methods to approximate those blocks as backups. In comparison to a conventional global and local redundancy-based approach, our proposed method achieves accuracy close to the original network while significantly reducing hardware overhead.

The contribution of this thesis is summarized in the tree diagram 1.7

1.7. Overall Reliability Flow

The uncertainty estimation and reduction methods proposed can be combined to improve the overall reliability of edge AI accelerators. However, the overall reliability flow for the proposed approaches differs depending on whether runtime monitoring and maintenance approaches or self-healing approaches are used. In runtime monitoring, the overall flow can also differ depending on whether periodic or concurrent monitoring is performed. Nevertheless, the reliability flow proposed in this thesis is end-to-end. That is, we propose optimizing for reliability across the entire stack, from the algorithm design to the hardware architecture.

However, regardless of the specific method used, the initial steps are the same. Specifically, baseline (fault-free) performance metrics are established in a cloud-based architecture. Then, if the performance is acceptable, the pre-trained model is mapped in the edge AI accelerator. The model can be trained using the proposed non-idealities-aware or self-healing training methods or a pre-trained model from MLaaS. Afterward, post-mapping testing is performed to detect faults and variations in the accelerator. If needed, reliability improvement approaches such as re-calibration, re-training, or hardware module replacement

Figure 1.7.: Summary of the contributions of this thesis.

are performed. In addition, new performance metrics are established and adjusted based on the current state of the model. Once these steps are completed, the model is deployed for inference.

1.7.1. Runtime Monitoring and Reliability Improvement

With runtime periodic monitoring, key indicators such as accuracy [41] and logits distribution [42] of the AI accelerator on test data are analyzed for fault detection. Specifically, the monitoring data are analyzed to identify any deviations from the baseline or expected behavior that may indicate faults or variations in the accelerator.

At first, the dataset for inference is applied to the AI accelerator, and predictive uncertainty is estimated. If there is uncertainty, system backup is activated, and a root cause analysis is performed to determine the underlying cause, i.e., whether the uncertainty is due to hardware faults or environmental factors.

In the case of functional uncertainty due to hardware faults, more sophisticated tests are done, e.g., for fault localization. Afterward, a reliability enhancement method is applied that involves actions based on the root cause analysis. Specifically, this could involve hardware-level maintenance such as replacing

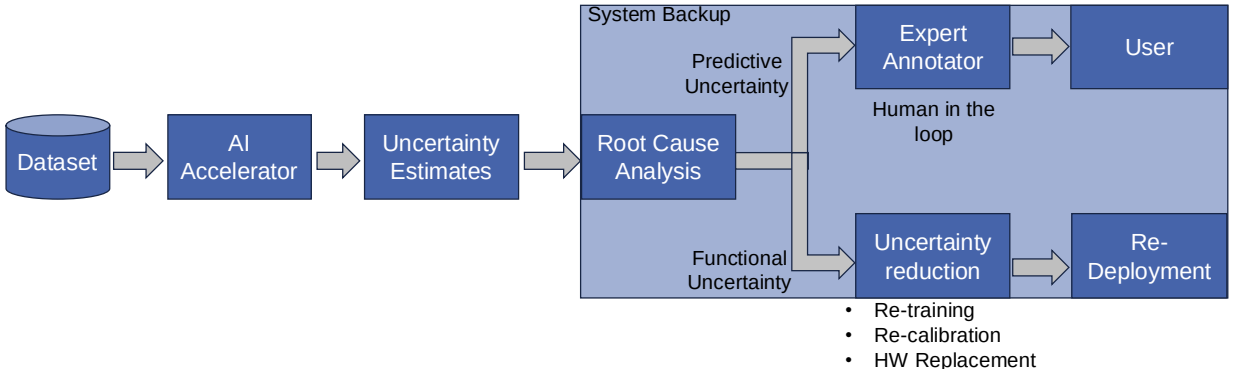


Figure 1.8.: The overall reliability flow for the edge AI accelerator using the runtime monitoring and maintenance methods proposed in this thesis.

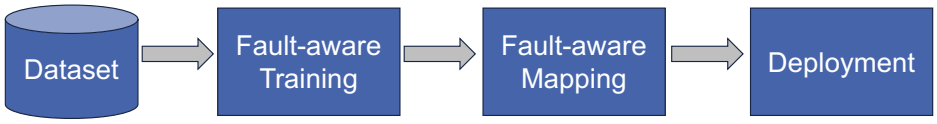


Figure 1.9.: Self-healing-based reliability flow for the edge AI accelerator proposed in this thesis.

faulty components, re-calibration of normalization layers, or algorithm-hardware solutions such as re-training and re-mapping the updated model parameters. After uncertainty reduction methods are applied, the functional uncertainty of the edge AI accelerator is re-estimated for verification. If the functional uncertainty is satisfactorily low, then the model resumes normal operation. Otherwise, a more sophisticated uncertainty reduction method or full hardware replacement may be required before the model can resume normal operation.

On the other hand, in the case of predictive uncertainty, input data is annotated by an expert annotator, for example, in a human-in-the-loop scenario, and the annotated label reaches the end user.

Conversely, with concurrent monitoring, key indicators are monitored concurrently with model predictions. In addition, the monitoring data is also analyzed concurrently. In the case of deviation from the expected behavior, the model prediction is discarded, and the input is reviewed by an expert. Therefore, it prevents a potentially harmful prediction from reaching the end user. Afterward, the application reliability enhancement method is the same as that of periodic monitoring. The overall reliability flow for the runtime monitoring and maintenance is shown in Fig. 1.8.

1.7.2. Self-Healing Approaches

Self-healing approaches, by design, allow for graceful degradation in accuracy. In self-healing approaches, the NN trained using a fault-aware algorithm and AI accelerator is also designed to be fault-tolerant. Therefore, the NN can be directly mapped for fault tolerance in the presence of faults. Therefore, runtime monitoring is typically not needed. However, periodic monitoring and maintenance methods discussed in the previous section can also be appended to a self-healing approach proposed in this thesis to improve overall reliability. The overall reliability flow with self-healing approaches is shown in Fig. 1.9.

1.8. Outline Of This Thesis

The background and related contributions are discussed in chapter 2. Following this, in chapter 3, the literature survey of related works and their drawbacks are discussed. Then, the proposed predictive uncertainty estimation approaches are presented in chapters 4 to 6. Afterward, approaches for estimating functional uncertainty of edge AI accelerators are presented in chapters 7 to 10. Subsequently, approaches

for uncertainty reduction are presented in chapters 11 to 12. Later, in Chapter 13, an approach for ensuring the continuous availability of the AI accelerator is presented. Finally, chapter 14 concludes this dissertation and discusses future research directions.

2. Background

2.1. Neural Network

Neural networks (NNs) are computational models inspired by the structure and operation of biological neural networks. NNs consist of multiple layers of neurons organized into a sinarithmetic a single output, and multiple hidden layers. The input layer does not perform any computation but only receives the input data. However, the hidden layers compute intermediate activations $\mathbf{z}$, and the output layer generates the final results $\hat{\mathbf{y}}$. The basic computation of a layer l consists of the weighted sum of inputs $\mathbf{x}$ and the element-wise addition of bias. Subsequently, a non-linear activation function $\phi(\cdot)$ is applied. The overall mathematical computation of the NNs is as follows:

$$\mathbf{z}^{(0)} = \mathbf{x}, \tag{2.1}$$

$$\mathbf{z}^{(l)} = \phi^{(l)}\left(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}\right), \quad l = 1, 2, \ldots, L-1, \tag{2.2}$$

$$\mathbf{z}^{L} = \mathbf{W}^{(L)}\mathbf{z}^{(L-1)} + \mathbf{b}^{(L)}, \tag{2.3}$$

$$\hat{\mathbf{y}} = \bar{\phi}\left(\mathbf{z}^{L}\right) \tag{2.4}$$

where, $\mathbf{W}$, $\mathbf{b}$, L, and ϕ represent the weight matrix, bias vector, the total number of layers, and activation function, respectively. The final output of the model z^L is called the *logits* of a model. Logit values represent un-normalized (row) outputs of a model before applying a task-specific final activation function ($\bar{\phi}$) to convert them into probabilities, e.g., the SoftMax function in classification or Sigmoid for semantic segmentation. The logit distribution of an NN is referred to as the distribution of these raw output values across different classes for a given input or batch of inputs. A logit distribution of a model can be fitted to a predefined distribution, such as Gaussian or uniform, with its specific parameters such as μ (mean) and σ (standard deviation).

Traditionally, NN topologies have a single "head," which refers to the last layer responsible for the final output. However, in modern deep learning paradigms, it is increasingly common to have task-specific heads [43].

2.2. Binary Neural Network

Usually, NNs have fixed-point weights and activations that require a larger memory size to store the trained weights and are computationally expensive. *Binary Neural Networks* (BNNs) use binary (+1 or −1) weights and activation functions during their inference by mapping positive values to +1 and negative values to −1 [44]. Hence, it requires only one bit to store a single trained weight and replace MAC with multiple XNOR and bit-counting operations [45]. By reducing the bit-width of the weights from a multi-bit fixed point to a single bit per weight, an NN layer can be directly mapped to a memristor-based crossbar that has only two stable states. Therefore, it effectively overcomes the limited stable state challenge of memristors. In addition, this can greatly reduce both computational power and storage requirements for memristor-based NN implementations with negligible performance penalties.

Relevant Research Direction Related to Scale vector

Most of the works in this thesis are focused on binary NN because of the benefits discussed above. Specifically, our goal is to estimate uncertainty, improve testability, and improve reliability in an efficient and scalable way.

2.2.1. Scale Vector In BNN

The scale vector $\boldsymbol{\alpha}$ is a crucial aspect of BNN to alleviate the loss of accuracy due to binarization [45]. Here, a real-valued vector multiplies the weighted sum, weights, or activations of a layer. The scale vector can be defined in two ways, such as analytically calculated values that scale the binarized weights and activations for each layer in [45] or can be learned via backpropagation similar to other parameters of the model [46].

In terms of the location of the scale vector, applying the scale to the weight matrix of each layer before the XNOR operation (as done in [47]) is possible in a CPU or GPU implementation, but may not be as feasible for a CIM architecture. This is because, depending on the shape of the scale vector, each neuron or channel will have a different scale factor, leading to different mapping strategy requirements for each neuron or channel. Similarly, input scaling is not feasible, as the inputs are directly converted to voltages and fed into the crossbar for computation.

Relevant Research Direction Related to Scale vector

In this thesis, we specially design the scale vector for uncertainty estimation [48, 49] and its application so that it can be implemented in the CIM architecture.

2.3. Normalization Approaches In Deep Learning

In modern deep learning topologies, normalization layers are essential to improve training stability, speed, convergence, and performance [50]. In general, the normalization layers standardize its input $\mathbf{z}$, as follows:

$$\text{BatchNorm}_{\gamma,\beta}(\mathbf{z}) = \frac{\mathbf{z} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \times \boldsymbol{\gamma} + \boldsymbol{\beta}. \tag{2.5}$$

where, the mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ are calculated across a specific dimension (batch, feature map, channel groups) depending on the type of normalization method. For instance, batch normalization [50] normalizes activations across a mini-batch, layer normalization [51] normalizes across all features of a single example, Instance Normalization [52] normalizes independently within each channel of a single example, and Group Normalization [53] normalizes across groups of channels. Furthermore, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters, and ϵ is a small constant for numerical stability.

Relevant Research Direction Related to Scale vector

One of the goals of this thesis is to reduce the cost of normalization during inference, specifically when the reliability of AI accelerators is centered around normalization layers. We also focused on a) modifying the batch mean and variance calculations, b) incorporating Dropout into normalization layers, c) efficiently re-calibrating the mean and standard deviation for fault tolerance, and d) approximating the batch normalization during inference, which allows completely removing computations in equation 2.5 during inference.

2.4. Regularization Methods

Overfitting is a typical problem in point estimate NNs, in which the neural network model performs extremely well on training data but not on inference data. To improve the generalization of the NN model to inference data, a regularization term is usually used, such as L_2 (ridge regression), applied to each parameter of the NNs. The overall learning objective of regularized NNs can be described as:

$$\mathcal{L} = \frac{1}{Q}\sum_{q=0}^{Q} E(y_q, \hat{y_q}) + \lambda \sum_{i=1}^{L}(|| \mathbf{W}_i ||_2^2 + || \mathbf{b}_i ||_2^2). \tag{2.6}$$

Where λ is the hyperparameter that controls the strength of regularization.

Stochastic regularization techniques such as Dropout [54] are also commonly used in NN models. Dropout adopts the model's output stochastically to perform regularization. Consequently, the loss becomes stochastic as well. The Dropout approach produces an average of the predictions of large ensembles of different neural networks in a computationally inexpensive way. During training, some neurons from the hidden layers are randomly omitted with a predefined probability. Dropout is applied by sampling binary vectors $\mathcal{M}_{\mathbf{i}}, i \in [1, \cdots L-1]$ of the same dimension as the bias vector $\mathbf{b}$. Each element of $\mathcal{M}_i$ is distributed according to a Bernoulli distribution with a probability $p = [0, 1]$. Therefore, Dropout can be described as $\mathcal{M}_{\mathbf{i}} \sim \text{Bernoulli}(1-p)$ and sets the given input $\mathbf{z}_i$ for a layer to zero $\mathbf{z}_i \odot \mathcal{M}$ with probability p. Where $\odot$ is the Hadamard product.

There have been other variants of Dropout that have different goals and capabilities. DropConnect [36], for instance, modifies the Dropout technique to operate on weights. The Gaussian Dropout variant [55] replaces dropped neurons with Gaussian noise, which can be interpreted as adding a measure of uncertainty to the Dropout procedure. On the other hand, spatial Dropout [56] removes entire feature maps from the convolutional layers, making the network resilient to loss of spatially correlated features.

2.4.1. Drawbacks for Edge AI-Accelerators

Although these methods are effective in their respective domains, they have drawbacks when it comes to uncertainty estimation and reduction in resource-constrained edge AI accelerators. Specifically, they a) offer limited inherent robustness to non-idealities of AI accelerators, b) they are not resource scalable, b) conventional dropout may require design time exploration to find optimal location and dropout rate, and c) conventional dropout is not applicable to certain parameter groups of a NN, e.g., scale vector and weights ($\boldsymbol{\gamma}$) of normalization layers. Otherwise, the loss of information occurs. That is, the signal flow becomes zero.

Consequently, in this thesis, we propose several novel Dropout approaches and learning objectives, uncertainty estimation, and reduction in resource-constrained edge AI accelerators.

2.5. Expectation of Uncertainty Estimates In Edge AI Accelerator

In deep learning, uncertainty estimation is crucial for evaluating the reliability and robustness of model predictions. It offers vital information about confidence in these predictions. This is especially crucial in supporting decision-making in safety-critical applications such as autonomous driving and automatic medical diagnostics.

A reliable uncertainty estimation method should demonstrate low uncertainty in data similar to what it has been trained on, in distribution (ID) data, and high uncertainty on unseen or OOD data. In a fine-grained method, an incorrect prediction should show high uncertainty, and a correct prediction should show low uncertainty.

Note that there is a difference between generalizing on the same data, i.e., inference accuracy and OOD data. Inference accuracy refers to prediction accuracy with data that have the same distribution as training data but are unseen during training, e.g., validation data. An ideal uncertainty estimation method, during inference, is expected to generalize well on the same data distribution and provide interpretable uncertainty estimates on OOD data.

2.6. Bayesian Neural Networks

Bayesian Neural Networks (BayNNs) take a fundamentally different approach to conventional NNs. Instead of finding a single best parameter vector $\boldsymbol{\theta}$, BayNNs consider a distribution of possible parameters, represented as $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$. In this case, the learning process corresponds to estimating the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ given the data $\mathcal{D}$.

This change in perspective provides a significant benefit: it enables us to estimate a distribution for our predictions, given by

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{\theta}. \tag{2.7}$$

Consequently, we are able to estimate not only the values of our predictions but also the associated uncertainty, which is frequently essential for practical applications, such as safety-critical applications, including autonomous driving, industrial robotics, and so on.

However, training a BNN is not as straightforward as training a conventional NN. This is because the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ cannot be computed directly.

In many scenarios, especially when dealing with high-dimensional and complex models like NNs, the exact computation of the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ is computationally intractable. This difficulty arises due to the need to calculate the integral in the denominator of Bayes' theorem, which is also known as the evidence or the marginal likelihood:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})\, d\boldsymbol{\theta} \tag{2.8}$$

This integral is over all possible values of $\boldsymbol{\theta}$, and in high dimensions, direct computation becomes practically impossible. That is why we usually cannot obtain the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ in closed form, and we resort to approximation techniques.

2.6.1. Variational Inference

Among the approximation techniques, Variational Inference (VI) is theoretically grounded, generally applicable, and computationally efficient [57]. In VI, a more computationally convenient variational distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta} \mid \mathcal{D})$ with parameters $\boldsymbol{\omega}$ is employed instead of the true posterior distribution.

Generally, the variational distribution $q_{\boldsymbol{\omega}(\boldsymbol{\theta})}$ is chosen to be a Gaussian with a diagonal covariance matrix, characterized by the mean $\boldsymbol{\mu}_\omega$ and variances $\boldsymbol{\sigma}_\omega^2$. The variational parameters are found by minimizing the Kullback-Leibler divergence $\mathrm{KL}\left(q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta} \mid \mathcal{D})\right)$ with respect to $\boldsymbol{\omega}$ [34]. Minimizing the KL divergence encourages the variational distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ to be close to $p(\boldsymbol{\theta} \mid \mathcal{D})$, leading to a good approximation of the true posterior distribution.

Once we have approximated $q_{\omega}(\boldsymbol{\theta})$, we can then make predictions $\mathbf{y}^*$ for given inputs $\mathbf{x}^*$ using a Monte Carlo approximation:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} p\left(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}^{(t)}\right) \quad \text{with} \quad \boldsymbol{\theta}^{(t)} \sim q_{\omega}(\boldsymbol{\theta} \mid \mathcal{D}). \tag{2.9}$$

Research Direction Related to VI

This thesis focuses on realizing the VI-based BayNN paradigm in a hardware-friendly manner, with an emphasis on devising a) a *Compute-in-Memory (CiM) accelerator-friendly approximation* for the posterior inference of $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$, b) novel BayNN topology for efficient sampling in CiM, and c) VI in binary neural networks, d) reducing memory and power consumption for resource-constrained edge AI applications, without compromising inference accuracy and quality of uncertainty estimates.

2.6.2. MC-Dropout as Bayesian Approximation Approximation

Gal et al. [35] provided mathematical groundwork showing, in the case of a variational distribution, that columns that are randomly set to zero with a probability p can also act as an approximation of the intractable posterior distribution. In addition, minimizing L_2 regularization is similar to minimizing the KL divergence.

The overall objective can be described as:

$$\mathcal{L} \propto \frac{1}{\tau Q} E_{p(\hat{\theta}|\mathcal{D})} + \sum_{i=1}^{L} \left(\frac{p_i \ell^2}{2\tau Q} \mid\mid \hat{\mathbf{W}}_i \mid\mid_2^2 + \frac{\ell^2}{2\tau N} \mid\mid \hat{b}_i \mid\mid_2^2\right). \tag{2.10}$$

The overall objective is similar to Equation 2.6 with ℓ as the prior length-scale that defines a more expressive prior, τ model precision. Therefore, sampling stochastic parameters $\hat{\theta}$ from the Bernoulli distribution is equivalent to the binary variables of the Dropout. Here, $\hat{\theta}$ summarizes the stochastic weight $\hat{\mathbf{W}}$ and bias $\hat{b}$ in Equation 2.10. Specifically, the Equation 2.10 is is approximated by

$$\mathrm{p}(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} \mathrm{p}(\mathbf{y}^*|\mathbf{x}^*, \hat{\boldsymbol{\theta}}, \mathcal{M}_t) \quad \text{with} \quad \mathcal{M}_t \sim \text{Bernoulli}(1-p). \tag{2.11}$$

Here, the entries of $\mathcal{M}_t$ are independently sampled from a Bernoulli distribution with (dropout) probability p.

Predictive performance can be determined by averaging the outputs of the T samples, see Fig. 2.1. In addition, the variance of the T samples can be used as a measure of uncertainty in the prediction. Therefore, uncertainty for an NN can be easily obtained by using Dropout during inference for an NN with floating point parameters θ, given that it was trained with Dropout and L_2 regularization.

Relevant Research Direction Related to MC-Dropout

This thesis focuses on hardware-software co-design of the MC-Dropout-based BayNN in Spintronics-based CiM architectures. Specifically, the emphasis is on devising a) MC-Dropout-based Bayesian binary neural networks, b) reducing the number of dropout modules using grouped and vector Dropout, c) novel Dropout-based Bayesian inference, d) reducing chip and power consumption related to Dropout modules,

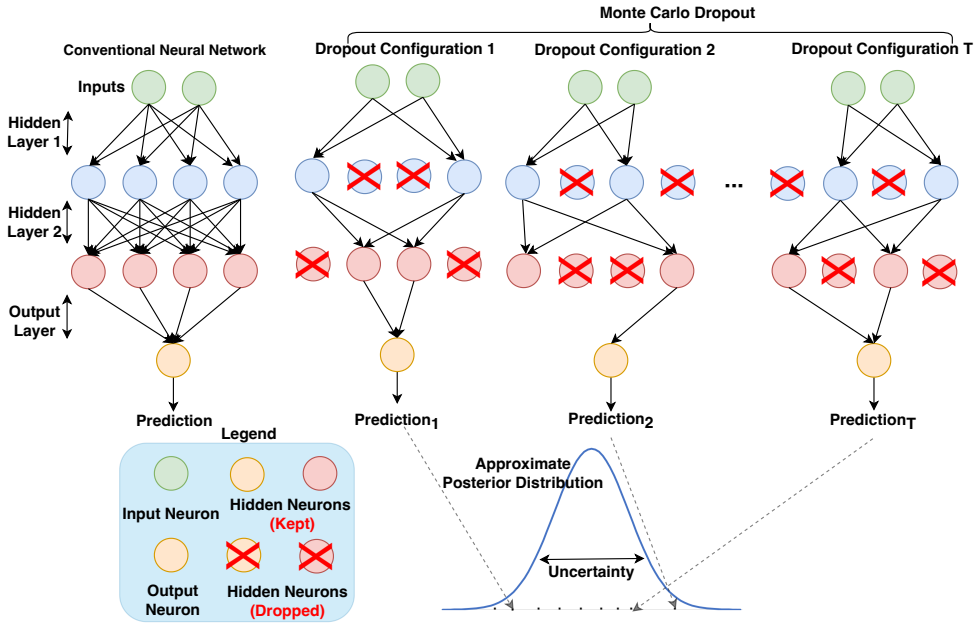


Figure 2.1.: Bayesian inference with MC-Dropout NN in comparison with conventional NN. In MC-Dropout, multiple forward passes are performed to obtain the posterior distribution. **It is recommended to view this figure in color.**

e) improving reliability via fault-tolerance of BayNN, and f) functionally online testing of such model, without compromising inference accuracy and quality of uncertainty estimates.

2.7. Model Ensemble For Uncertainty Estimation

The model ensemble method is one of the most popular and highly successful approaches for uncertainty estimation due to its high inference accuracy and quality uncertainty estimates.

The ensemble of models involves combining predictions from multiple individual models (see Fig. 6.1(a)) to improve overall performance and estimate uncertainty. During training, M models are trained independently or collaboratively using techniques such as bagging or boosting. These models can be trained with different architectures, initializations, or subsets of data to encourage diversity. During inference, predictions from different models are aggregated using methods such as averaging or weighted averaging to obtain the final prediction. Since training, storage, and processing of M full models are required, the hardware cost, e.g., memory, latency, and power consumption, is a concern for edge AI accelerators with limited resources.

Relevant Research Direction Related to Model Ensembles

In this thesis, the aim is to reduce the cost of the model ensemble approach via designing a novel approach that a) is suitable for existing AI accelerator architecture, b) is suitable for resource-constrained edge AI devices, c) is scalable, d) is capable of single-shot training and inference, e) does not compromise inference accuracy, and f) provides high-quality uncertainty estimates.

2.8. Memristor Technologies

Emerging resistive nonvolatile memories (NVMs) such as two-terminal phase change memory (PCM) [58], resistive random access memory (RRAM) [59], and spin-transfer torque magnetic random access memory

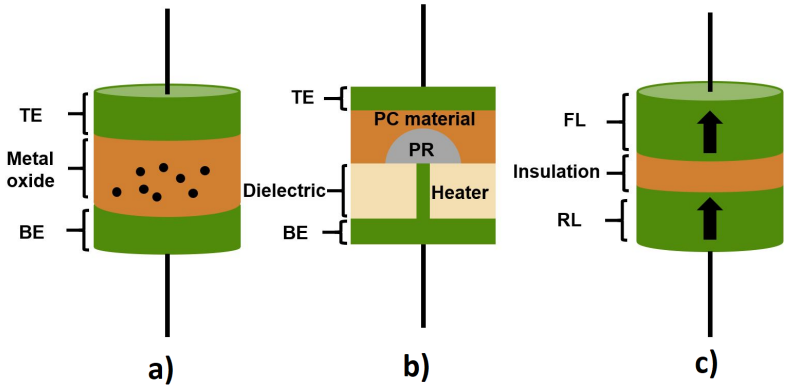


Figure 2.2.: Cross-section of a) RRAM cell, b) PCM cell with a programmable region (PR), top electrode (TE), bottom electrode (BE), and phase change (PC) material, and c) the MTJ device in parallel (*P*) state (both the reference (RL) and the free layers (FL) are aligned). **It is recommended to view this figure in color.**

(STT-MRAM) [60] offer the benefit of high speed, low leakage power, scalability, acceptable endurance, and CMOS compatibility. The cross sections of the RRAM and PCM cells are shown in Figs. 2.2(a) and (b), respectively. They have two resistance states: Low-Resistance State (LRS) and a High-Resistance State (HRS). The data is represented and stored as a resistance state, and the states can be changed by applying a proper set/reset voltage.

2.8.1. Spintronic Technology

Magnetic random access memory (MRAM) has gained significant attention due to its potential use in CiM architectures [61]. The Magnetic Tunnel Junction (MTJ) is composed of two ferromagnetic layers: a reference (RL) and a free layer (FL). A thin oxide insulation layer is sandwiched between the two magnetic layers, as shown in Figure 2.2 (c). The reference layer has its magnetic orientation fixed, but the free layer can change its magnetic orientation according to the direction of the current applied through the MTJ.

The freely magnetized layer can be reversed mainly through two writing mechanisms: Spin Transfer Torque (STT) and Spin-Orbit Torque (SOT). The resistance state of the MTJ depends on the magnetic orientation of the two layers. It is in the LRS state when the magnetic orientation of the layers is aligned (P) and is in the HRS state when the orientation is not aligned (AP).

STT-MRAM has common read and write paths, but SOT-MRAM has separate read and write paths. When the reading current is increased, the stability of the STT-MRAM is degraded, e.g., due to read disturb error, which leads to a higher error rate. In contrast, the reading reliability of SOT-MRAM is significantly better due to separate read and write paths. SOT-MRAM is made up of an MTJ placed on a heavy metal layer.

2.8.2. RRAM Technology

An RRAM device consists of a resistive layer sandwiched between two electrodes. In addition, typically, transition metal oxides, such as HfO_x or NbO_x, are used for the resistive layers. During programming, the device conductance is increased with the “SET” operation, whereas the conductance is decreased with the “RESET” operation. Generally, RRAM can be categorized into analog and digital types. In analog RRAM, device resistances can be programmed to any value between the highest resistance state (HRS) and the lowest resistance state (LRS). On the contrary, the binary RRAM has stable HRS and LRS and behaves as a normal memory device.

Table 2.1.: Showing XNOR operation for BNN implementation.

Input/Activation	weight	Bitwise XNOR	Effective Resistance
+1	+1	+1	LRS
−1	−1		
+1	−1	-1	HRS
−1	+1		

2.8.3. PCM Technology

The PCM device is made of chalcogenide material and is placed between two electrodes. Initially, the chalcogenide is in the crystalline state, but its state can be changed by a melt-quench process. The electrical resistance of the material in the crystalline and amorphous states differs significantly, enabling the PCM device to store information based on its phase. A high resistance state is achieved by applying a strong current (RESET pulse) through a narrow bottom electrode, creating a mushroom-shaped amorphous region that blocks the electrical pathway. This amorphous region can be gradually turned back into a crystalline state by raising the temperature of the device using current pulses (SET pulses) of appropriate strength. The ability to gradually modulate the conductance of the device allows PCM to store weights.

2.9. AI Accelerator Architectures

2.9.1. Memristor-based Computation-in-Memory Architectures

Weight Mapping Strategies A critical step in performing the computation inside the CiM architecture is to map the different layers of the NN to crossbar arrays. Standard NNs contain mainly fully connected (FC) layers and convolutional layers. Although the mapping of FC layers is straightforward in a crossbar array, since the shape of the weight matrices is 2D ($\mathbb{R}^{m\times n}$), the mapping of convolutional layers is challenging due to their 4D shapes ($\mathbb{R}^{K_h\times K_w\times C_{in}\times C_{out}}$). Here, $K_h \times K_w$ denotes the shape of the kernels, C_{in}, and C_{out} represents the number of input and output channels, respectively. Implementing convolutional layers requires the implementation of multiple kernels with different shapes and sizes.

There are two popular mapping strategies for mapping the convolutional layer. In the mapping strategy ①, each kernel of shape $K_h \times K_w \times C_{in}$ is unrolled to a column of the crossbar [62]. On the other hand, in the mapping strategy ②, each kernel is mapped to $K_h \times K_w$ smaller crossbars with a shape of $C_{in} \times C_{out}$ [63].

The trained parameters θ of the NN are mapped to the memristor-based crossbar arrays by encoding each bit as the memristor conductance. Each memristor cell in the crossbar array represents a single-bit (0 or 1) by programming them to either a high or low conductance level (G_{off} or G_{on}).

Computation in Memory Operation The memristor cells are arranged in a crossbar array as shown in Fig. 2.3 to perform the MAC operation required for the inference stage of the multi-bit quantized NN. The crossbar structure allows the weighted sum operation to be carried out directly in the memory at a constant $O(1)$ time without any data movement between the processing element and the memory.

In BNN, the crossbar structure is designed differently. Specifically, the XNOR-bitcell design proposed in [64] is used mainly for the implementation of BNN in this thesis. Due to its complementary bit-cell design, the same effective resistance/conductance appears in the bit-line depending on input/activation and weight combination. That is, $+1 \oplus +1$ and $-1 \oplus -1$ result in an LRS in the bit-line. Here, $\oplus$ represents the XNOR operation, and the overall XNOR computation required for BNN is shown in Table 2.1.

The input vector $\mathbf{x}$ for the inference is converted into continuous voltages and streamed into the crossbar array. Multiple word-lines m_{wl} of the crossbar are activated at the same time in a single step (s). The resulting current that flows into the bit-line of the crossbar, referred to as the partial sum current (I^s_{ps}), is

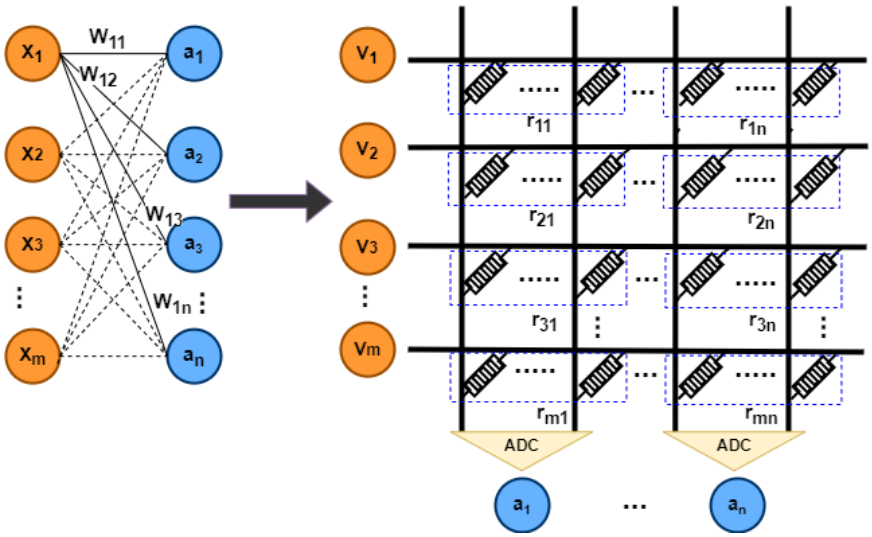


Figure 2.3.: Graphical demonstration of the mapping of a fully-connected layer to a memristive crossbar array of size $m \times n$.

sensed by a sensing circuit and digitizes them. The partial sum current represents the partial outcome of the MAC operation, and it is accumulated to obtain the final MAC results of a layer after activating all the word-lines. Therefore, even though the activation of BNN is binary, the partial sum current needs to be sensed with high precision (error-free). The addition of bias, batch normalization, and non-linear activation operations are performed following that.

Data converter circuits such as the digital-to-analog converter (DAC) and the analog-to-digital converter (ADC) are used to apply the input vector $\mathbf{x}$ and sense the bit-line currents of the crossbar. However, in BNN, 1-bit sense amplifiers (SA) can also be used to sense the bit-line currents using the input-splitting approach proposed in [65]. Their approach splits the weight matrix into several crossbar arrays, which are connected to an output crossbar array.

Note that the data converter circuits are also subject to variations. However, they can be designed so that they are robust against variations [66].

Research Direction Related to CiM Architectures

This thesis offers an alternative method for using SA for sensing bit-line current in BNN that is proven to be robust to manufacturing and runtime thermal variations of the memristive device (STT-MRAM) [67].

Furthermore, several novel CiM architectures are proposed.

2.9.2. Digital AI Accelerators

In digital AI accelerators, highly parallel computing paradigms are used very frequently, including both temporal and spatial architectures. In central processing engines (CPUs) or graphics processing units (GPUs), temporal architectures are mostly employed, but application-specific integrated circuit (ASIC) and field programmable gate array (FPGA)-based designs commonly employ spatial architectures.

Specifically, temporal architectures use a variety of methods to improve parallelism, such as vectors (single instruction, multiple data (SIMD)) or parallel threads (single instruction, multiple threads(SIMT)). Furthermore, they use a large number of arithmetic logic units (ALUs) with centralized control. However, although the ALUs cannot communicate directly with each other, they can fetch data from the memory hierarchy.

On the other hand, dataflow processing, i.e., the ALUs form a processing chain, is employed in spatial architectures as they can directly transfer data from one to another. In some cases, each ALU is equipped with its own control logic and local memory, also called a processing engine (PE). To reduce the energy cost due to data movement, several levels of local memory hierarchy are introduced in spatial architectures [68].

2.10. Failure Mechanisms in Memristor-based CiM

Memristive devices suffer from various defects, variations, and non-idealities [69, 70, 71, 72, 73, 74, 75, 76]. As a result, the post-mapping NN model parameters deviate from their original values. In this section, we give details about some common defects of memristor devices and their respective fault modeling.

2.10.1. Permanent Faults

We refer to the faults that permanently change the conductance state of the memristor cells and cannot be programmed to the desired resistance/ conductance state to encode NN parameters as permanent faults. A cell with permanent faults can not be refreshed back to its original fault-free values. We discuss some of the common permanent fault models.

In spintronic devices, the magnetic coupling phenomenon pertains to stray magnetic fields from the ferromagnetic layers of neighboring MTJs or within an MTJ cell and affects the stability of magnetization in the FL of MTJs.

Stuck-at faults The conductance of the memristor cell can be stuck at either high conductance stuck-at-G_{off} or low conductance stuck-at-G_{on} due to repeated reading (limited endurance) and fabrication defects. Therefore, it corresponds to the particular bit of the NN parameter being either stuck-at-0 (stuck-at-(-1) in BNN) or stuck-at-1, depending on the weight encoding. In this thesis, stuck-at-G_{off} and stuck-at-G_{on} correspond to stuck-at-0 (stuck-at-(-1) and stuck-at-1, respectively, based on the encoding discussed later. Defects such as stuck-open or short can be modeled as opens and shots between different nodes in the memristor cell. Stuck-open and stuck-short defects result in a change in the conductance of a memristor cell from its normal range and imply that the memristor cell is either open or short. The effect of stuck-short can cause a large current through the corresponding bit-line, and it can be harmful to the circuit. Therefore, the parameters of the memristor-based NN implementation will change from their original values, and the parameter bit change can be modeled the same as stuck-at-G_{off} and stuck-at-G_{on}.[69, 70]

Manufacturing defects in Spintronics devices include front-end-of-line (FEOL) defects such as semiconductor impurities, crystal imperfections, and pinholes in gate oxides, and back-end-of-line (BEOL) defects such as pinholes in MgO barriers, sidewall re-depositions, and magnetic layer corrosion.

2.10.2. Soft faults

In this thesis, we refer to soft faults as faults that temporarily change the conductance state of memristor cells but can still perturb the NN parameters. However, faulty memristor cells can be refreshed back to their original values. We discuss some of the common soft fault models.

2.10.2.1. Read/write disturbance

During memristor reading (inference) and writing (parameter mapping), both read and write current can affect other memristor cells sharing the same bit-line in the crossbar array. Such faults can cause accidental switching of the conductance states of the memristors during the read operation (inference). Also, the write disturbance fault affects the stored data (NN parameters) in memristor cells [70, 73].

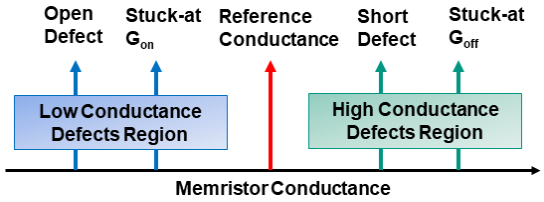


Figure 2.4.: Demonstration of the low and high conductance regions of the Memristor with open/short and stuck-at defects.

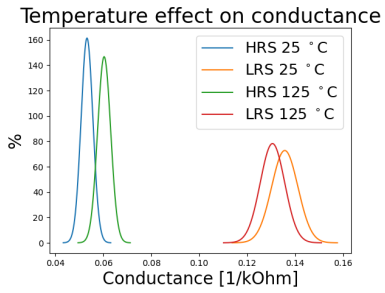


Figure 2.5.: Conductance variations of MTJs showing device-to-device and thermal variations. **It is recommended to view this figure in color.**

2.10.2.2. Slow-Write Fault

During the NN parameter mapping, the write delay of the defective memristor cells may be longer, which is referred to as a slow-write fault. In RRAM, slow-write faults can arise from repeated write operations. Switching in MTJ and PCM is inherently stochastic. STT-switching stochasticity is influenced by variability in incubation delay and actual switching time, leading to transient faults during write operations.

Therefore, the write delay is also non-deterministic even when the environmental factor is the same. A write failure can occur if the MTJ does not switch before a specified time or the switching pulse is truncated before the switching operation is performed [72, 71].

2.10.2.3. Retention Failures

Retention faults are a major reliability issue. Ideally, after a memristor cell is written, its content is expected to be stored until the next write operation or the expected operating time of the device (typically 10 years). However, due to external influences, the cell may lose its datum, or the written resistance drifts in multi-level memristor cells. Additionally, most memristors have an asymmetrical flip behavior, making it more probable for an NVM cell to change to a certain state over time [77, 78].

In the case of the MTJ, the energy barrier from the *HRS* to the *LRS* is lower than the barrier from *LRS* to *HRS*, making it more probable for an MTJ to switch from the *HRS* to the *LRS* than the other way around [79]. Additionally, increasing the temperature lowers the effective energy barrier, which in turn increases the possibility of retention faults [80].

Retention faults in STT-MRAM happen suddenly and are considered a stochastic process due to external influences, e.g., thermal noise [81]. The probability of retention faults (P_{RF}) depends on the thermal stability factor Δ of the MTJs and can be expressed by

$$P_{RF} = 1 - \exp\left(\frac{-t}{t_0 \times \exp \Delta}\right), \tag{2.12}$$

where t_0 is the time constant of value 1 ns, Δ is the thermal stability factor, exp is the exponential operation, and t is the observed time interval [82].

In general, the lower the thermal stability factor Δ of the MTJs, the higher the uni-directional ($HRS \longrightarrow LRS$) switching probability P_{RF}, and the more uni-directional faults are expected. Higher uni-directional fault rates can lead to higher degradation in inference accuracy over the expected device operational time t_{end}, or the expected time before the next weight matrix update [29].

2.10.3. Variations

Due to manufacturing variations, the conductance of the memristor device varies device-to-device, showing a distribution rather than a fixed value as shown in Fig. 2.5 for the MTJ device. Therefore, the accumulated current sum of the bit-line of the crossbar also varies and can be represented with a statistical distribution. Furthermore, the conductance of the memristive device can dynamically fluctuate over time as a result of temperature. As a result, the sensing margin of accumulated current reduces, and the overlapping region between the conductance states grows. This can lead to sensing the wrong value.

In Spintronics devices, extreme process variations lead to significant deviations in key MTJ parameters such as magnetic anisotropy, saturation magnetization, tunnel magnetoresistance (TMR) ratio, and cross-sectional area, affecting both MTJs and transistors. Furthermore, thermal fluctuation as a result of online temperature variations can significantly impact the magnetization reversal process in MTJs, causing retention and read decision faults. Due to these faults, the conductance state of a Spintronics device can suddenly switch to another state.

2.10.4. Failure Mechanisms of Buffer Memories

In memristor-based CiM, frequently updated results, such as intermediate activations, are stored in the register, flip-flops, or SRAM memories. In SRAM, permanent open and short faults can occur due to bridging defects that create an unwanted current that leads to a path between two nodes in the cell. Similarly, resistive open defects lead to an increase in resistance of existing paths within the cell [83]. Furthermore, transient faults can occur as a result of temperature and voltage fluctuations, as well as radiation particles that strike memory cells.

2.10.5. Failure Mechanisms of Dropout Modules of CiM

Several spintronics-based dropout modules are designed by our technology partners for the BayNNs proposed in this thesis [84, 85, 86, 48]. Due to the permanent and transient faults in the Spintronics device discussed before, the Dropout module can cause a word-line or a group of word-lines in the crossbar to be constantly active or inactive, or change the number of inactive word-lines. Furthermore, Dropout probability is highly sensitive to the switching current. Therefore, a small fluctuation in the switching current can cause variations in the Dropout probability.

2.10.6. Linear Block Error Correction Coding

Error correction coding (ECC) finds widespread use in electronic systems to ensure the protection of memory and computing elements [87, 88]. Linear block codes are a class of ECCs that operate on blocks of data [89, 88]. The linear block code can be defined as (n, k) code, where k is the length of data bits (u), n is the length of the codeword ($\bar{c}$), and $\bar{p} = n - k$ is a number of redundant parity bits. The k-bit data can be converted into a n-bit codeword using the ECC-specific generator matrix $\bar{G}$ as follows: $[\bar{c}]_{1\times n} = [u]_{1\times k} \times [\bar{G}]_{k\times n}$. Similarly, the syndrome (sy) computation can be done using the parity check matrix (H) to locate and correct the error in the received codeword (r), as follows:$[s]_{1\times n-k} = [r]_{1\times n} \times [H]^T_{n-k\times n}$.

2.10.6.1. Hamming Code

Hamming code [90] belongs to the class of single-bit error correction linear block codes. Hamming code enables the error correction capability by padding $\bar{p}$ parity bits to k bits of the data word and satisfies the following inequality: $2^{\bar{p}} \geq k + \bar{p} + 1$. Additionally, Hamming code can also be used as single error correction double error detection (SEC-DED) code by extending one extra parity bit.

2.10.6.2. BCH Code

The BCH codes [91] form a class of multi-bit error-correcting linear cyclic block codes. For any positive integers $m \geq 3$ and $t < 2^{m-1}$, there exists a binary BCH code with codeword length $n = 2^m - 1$, number of parity $\bar{p} \leq mt$ and $d_{min} \geq 2t + 1$, where d_{min} is the minimum Hamming distance, and t is the error correction capability. The detection capability of the BCH code can also be increased by extending one parity bit, thereby enabling them to correct up to t errors and detect up to $t + 1$ errors.

3. Related Works

The contributions made by this thesis are an attractive alternative to a range of previously published literature, reaching from the theoretical research of uncertainty in deep learning and NN training to the more hardware-oriented field of edge AI acceleration, CiM architectures, and reliability. The ways in which we advanced the fields of uncertainty estimation and reduction of resource constraint edge AI accelerator can be roughly categorized into two groups: algorithmic contributions and hardware centric contributions. This chapter presents an overview of the most relevant literature.

3.1. Uncertainty Estimation

3.1.1. Hardware Implementation of BayNN

Prior to the work done in this thesis, several studies have been conducted on hardware solutions for Bayesian and binary neural networks (separately). To the best of our knowledge, the combination of Bayesian and binary neural networks was not performed for uncertainty estimation, especially in CiM architectures.

Nevertheless, the general trend in existing prior works was to use implementations based on CMOS technology, such as graphics processing unit (GPU) platforms, for inference and training [92]. Some other studies favored the use of field programmable gate arrays (FPGAs) or application-specific integrated circuit (ASIC) solutions [93, 94, 95, 96, 97]. However, these solutions suffer from excessive power consumption due to data transfer between the memory and the core unit [98] and may be restricted when used with larger datasets, i.e., **not scalable**.

There are also several CiM-based implementations that exist. Specifically, the technique described in [99] involves a CIM implementation in which the crossbar arrays store the variance parameters and stochastic resistive RRAM devices are used to sample the probability distribution at the input of the array. This approach requires a single random element for each input, which is not very energy-efficient and scalable. In contrast, work [100] takes advantage of the non-idealities of RRAM devices to apply Bayesian learning. The research in [101] showed the application of a set of resistive crossbar arrays to store probabilistic weights to execute BayNN. In [102], crossbar arrays were used to construct Bayesian neural networks with the help of low-barrier MTJs, resulting in a significant decrease in energy consumption. Despite the fact that memories with low-energy barriers are used, they have endurance limitations that can eventually have an impact on the precision of the CIM engine. The paper in [103] presented an alternative implementation with MRAM-based crossbar arrays that can represent mean and variance. However, this approach required considerable pre-processing to encode the mean and variance in the crossbars.

In this thesis, the main aim related to Bayesian neural networks is to binarize them and implement them in Spintronics-based CiM architectures. Therefore, the benefits of binary neural networks, Bayesian neural networks, Spintronics devices, and the CiM architecture can be attained in a single package. Each of our methods [49, 84, 85, 104, 86, 48] optimizes the device level up to the algorithmic level for a highly efficient solution.

3.1.2. Variational Inference

Similarly, there are several variational inference-based approaches in the literature. Bayes by Backpropagation (BB-BackProp) in [34] proposed a backpropagation-compatible algorithm and is one of the most popular and efficient methods. This algorithm enables the learning of a probability distribution for the weights of a NN.

However, implementing the probability distribution and sampling from it during forward propagation in CiM hardware poses challenges. In this thesis, we aim to address this problem. Furthermore, since the parameters or activations are a distribution, VI on binary neural networks is infeasible. In this thesis, we also aim to address this problem.

3.1.3. Monte Carlo Sampling-based Approaches

On the algorithmic side, there are multiple approaches for estimating the uncertainty of NNs that extend the concept of MC-Dropout [35]. For example, MC-DropConnect [36] applies dropout to weight elements in a way that is impractical for CiM architectures. This because turning down individual memristor element is challenging or requires novel crossbar structure that requires manufacturing. On the other hand, work [105] proposed the MC-Batchnorm that uses the batch normalization layer for uncertainty estimation. Their approach requires passing a randomly sampled mini-batch from the training data through the NN and recalculating batch statistics. However, a drawback of this method is the requirement to store the training datasets on the hardware.

To reiterate, this thesis also proposed several extensions of the MC-Dropout approaches [48, 84, 86, 85] using the novel dropout approaches that act as an alternative to the original dropout in MC-Dropout. The main focus has been on reducing hardware costs and improving fault tolerance without (noticeably) compromising the accuracy or quality of uncertainty estimates.

3.1.4. Model Ensemble

The ensemble of models has been extensively studied to improve the performance of the model [106, 107, 108]. Even in this case, there are several methods to reduce the cost of inference. For example, the work in [109] proposed a model compression technique to compress large and complex models into smaller and faster ones. Similarly, work [110] introduced the knowledge distillation method, which distills model ensembles into a single neural network.

Since ensembles require training M models, several studies aim to reduce their cost at training time. For example, [111] proposed the Snapshot ensemble method, which encourages a single model to visit multiple local minima by training it using cyclic learning rates [112]. This method encourages the exploration of numerous local minima, which are then used as ensemble members.

Furthermore, to reduce inference time computational and memory overhead in ensemble methods, several studies also exist. In the context of model ensembles, Monte Carlo dropout (MC-dropout) can be interpreted as "implicit" ensembles that can create an exponential number of weight-sharing sub-networks for uncertainty estimates [35]. Although MC-dropout requires training and storage of a single model, inference involves M forward passes through a dropout-enabled network. Here, M varies with tasks, and the topology can be as large as 94 even on a small (six-layer) fully convolutional network [36]. Furthermore, MC-dropout has sampling latency and chip area overhead for the dropout module implementation, as shown in other contributions of this thesis [85, 86, 48]. To reduce inference latency, the work [113] proposed to ensemble only deeper convolutional layers while the shared backbone is computed only once and cached. However, in convolutional NNs (CNNs), deeper convolutional layers have significantly larger parameter counts than other layers, as shown in Fig. 6.2 (a). Also, this approach only works if dropout is applied only to deeper convolutional layers rather than to all layers. In contrast, the BatchEnsemble [114] approach also shares

weights but introduces two sets of M rank-1 matrices to generate M ensemble members. Their approach is not scalable to the AI accelerator architecture, which does not allow batch processing. Additionally, it introduces additional computation at the input and output of a layer, as shown in Fig. 6.1 (b).

In contrast, the low-cost and scalable ensemble approach proposed in this thesis [115] aims to optimize performance, training, and inference costs collectively with AI accelerator architectures in mind while providing quality uncertainty estimates.

3.2. Testing NNs in Edge AI Accelerators

There are hardware and algorithmic solutions to test AI edge accelerators that implement NNs. Some works target specific modules of AI accelerators or spiking neural networks. They are discussed below.

3.2.1. Hardware-based Solutions

Conventional hardware-based solutions such as March-based algorithms have been proposed to test memristor-based crossbar arrays [116]. March algorithms serially program and read the memristor cells under test to a specified conductance level to detect faults. However, a march-based algorithm is not practical for the memristor-mapped NN applications as they have a significantly large number of memory cells and will require a large test time. Also, the memristive cells have to be set to all possible levels (for multi-level cells), further exacerbating test time. Also, works propose to add sensors. Such as the work presented in [39], which proposes monitoring the dynamic power consumption of crossbar arrays to detect faults. To achieve this, an adder tree is implemented to monitor the dynamic power consumption continuously. However, it has added hardware overhead in terms of chip area and power consumption as well as security risk.

3.2.2. Algorithm-based Testing Approaches

Alternatively, several works focus on algorithmic methods to generate test vectors and online testing methods for AI accelerators. Specifically, a few works have focused on a deviation in the inference accuracy of either original training data or synthetic testing data in the presence of faults to detect faults [37, 117, 118]. Synthetic testing data are generated using adversarial examples in [117], watermarking the training data and re-training the NN on the testing data to create a backdoor in [37]. On the other hand, the work in [118] proposed back-propagating to the input image and utilising the gradient of the input image as the standalone testing data or combining it with training data as a perturbation, similar to [117] that uses the fast gradient sign method, an adversarial input generation method. Furthermore, Open Set Recognition (OSR) methods have been studied in work [119] for detecting single permanent faults affecting memory cells of NN accelerator architectures. However, they offer lower fault coverage, have higher false positive cases, and are only suitable for image classifications. Also, its adaptability to BNNs, multiple faults, or faults in the buffer memory that stores intermediate activations was not explored.

Although such methods are efficient at detecting deviation, they generally require a large amount of testing data and on-chip storage (depending on the availability of on-chip retraining data), and some methods require an invasive test generation procedure. Also, the performance of backdooring when common data-augmentation techniques, such as corner padding and center-cropping, are used is not established since data-augmentation can either partially or completely remove the watermarks. In addition, watermarking relies on the translation invariance feature of CNN to achieve high accuracy on the test dataset and similar performance on the original task. However, since MLPs are not translation invariant, their method may not work for MLPs.

3.2.3. Test Specific Modules of AI Accelerator

Another group of work aims to test specific modules of AI accelerators. For example, work by A Ruospo et al. [120] proposed testing floating-point multipliers of GPUs with image test libraries. Their experimental evaluation shows that the two image test libraries developed for two different CNNs (ResNet20 and DenseNet121), are able to find a 6-image image test libraries that could achieve about 95% coverage for stuck-at-faults on the GPU's multipliers.

In general, these kinds of approaches are more aligned with structural testing that targets specific hardware units of specific AI accelerators (e.g., GPUs) than functional testing of the entire NN model that is AI hardware-agnostic. Furthermore, some of these approaches are only targeted for soft faults [120], and are not targeted at permanent fault detection. Thus, these approaches may not be applicable to faults that occur in other locations of AI accelerators. For example, in the memory units that store MAC results or Rectified Linear Unit (ReLU) activation, e.g., dead ReLU.

3.2.4. Summary of the Gaps in the Existing Literature

The drawbacks of existing works (not applicable to all studies at the same time) can be summarized as follows: **a)** an invasive method of test generation, **b)** may not be scalable to different types of NN, **c)** targeted for specific modules of an AI accelerator, **d)** may not be scalable to different AI hardware, that is, not AI hardware agnostic **e)** requires large testing queries, **f)** has a significant storage overhead that sometimes increases with dataset size, and **g)** cannot always achieve high fault coverage.

3.3. Uncertainty Reduction

3.3.1. Variation Robustness

Improving the robustness of CiM architectures against process and temperature variations is a growing field of research. Several works have been proposed to mitigate the impact of process and temperature variations from the algorithmic level to hardware realizations by considering various memristor-based NN implementations.

For ReRAM and PCM eNVM technologies, works [24, 121, 122] have proposed runtime array-level column swapping techniques to change the mapping scheme and deal with the temperature variation. These methods swap important weights from areas with higher temperatures to those with lower temperatures. However, monitoring the temperature of the memristor-based crossbar array at run-time is expensive and requires additional hardware for fine-grained temperature sensing. When the temperature profile of the crossbar changes, extra effort is required for the re-mapping. Furthermore, once all cells in the crossbar reach an equivalent temperature, frequent and costly refreshing will be required.

The work in [123] shows a training algorithm that can deal with both the process and the temperature variation for the operating temperature range of the ReRAM device. It models the impact of temperature on the mean and standard deviation of the distribution of memristor cells and feeds them into the training algorithm as noise. Their approach trains several batch normalization layers for different operating temperature ranges. However, this method requires extra circuitry to switch between batch normalization layers at run-time and online temperature sensing. Furthermore, additional normalization layers have some memory overhead.

In [124] a temperature compensation technique is presented to mitigate the impact of temperature variations for PCM technology. This method modifies the activation calculation in hardware and requires extra processing during the activation calculation. Moreover, a compensation function is implemented as a

lookup table in hardware, which results in memory overhead. Moreover, the decoding process requires additional circuitry.

Other methods proposed robustness against process variation only that cannot mitigate run-time temperature variations. For example, the work in [125, 126, 127] proposed methods for process variation in ReRAM and STT-MRAM technologies, but their method cannot mitigate run-time temperature variations. Furthermore, the [125] change of the bit-cell design of STT-MRAM to 2T2R cells imposes a chip area overhead for the additional transistor and memristor cell.

3.3.2. Per-device Re-calibration For Variation-Tolerance

To overcome process and thermal variations, two previous studies proposed the online re-calibration of batch normalization. The authors of [30] proposed a generalized training algorithm to counteract the process variations of PCM technology and an online batch normalization parameter re-calibration technique similar to the work in [128] for ReRAM technology. However, their method requires large re-calibration dataset storage, which has a significant memory overhead. Furthermore, existing solutions [30, 128] have only been assessed against variation and full-precision NN.

Instead, the focus of this thesis is to reduce the overall re-calibration overhead via compacting re-calibration dataset using an automatic test generation method, approximating batch normalization, and partially re-calibrating. Furthermore, we adopt this concept for permanent and soft fault tolerance in binary neural networks.

3.3.3. Memory Scrubbing

Traditionally, memory scrubbing, periodic correction of corrupted data in memory, is done with an error correcting code (ECC) [129, 130]. In ECC-based scrubbing, several check bits are added to each memory word. Those words are periodically read, checked for errors, and written back with the corrected data in case an error is detected [131]. Since ECCs have an error detection and correction limit, they require frequent checks to prevent error accumulation, which interrupts the system's regular operation. A high storage overhead for the redundant check bits is needed to increase the correction capabilities. Specifically, for in-memory computing for AI acceleration, ECC-based scrubbing is impractical, as data encoding and memory access patterns with multiple rows at once do not benefit from word-line correction access.

In DRAM, memory cells are periodically refreshed to mitigate retention faults. Many works have been proposed to address retention faults in DRRM technology [132, 133, 134]. However, employing DRAM-style refresh to mitigate retention faults in memristors is challenging and not effective, where retention faults are not decay-based in some memristor technology and are temperature dependent. Furthermore, DRAM-style refresh will just result in reading the corrupted memory-cell content and writing it back [82].

Furthermore, some work has also been done for memristor technologies. Specifically, error correction checksum-based error correction is proposed for RRAM-based crossbars in [135]. However, memory overhead and power consumption do not scale well with the size of the crossbar. In [136], an ECC-based scrubbing technique for STT-MRAM is proposed, which has a 12.5% storage overhead. An adaptive scrubbing technique to mitigate retention faults in the *cache based* on STT-MRAM is proposed in [82]. They grouped the memory cells based on their retention time and adjusted the scrubbing interval with respect to the operating temperature. A training adaptation to reduce the number of *HRS* in the crossbar so that the number of uni-directional switching can be reduced is proposed in [29]. They also proposed a hybrid crossbar array with mixed retention cells to mitigate retention faults. Unfortunately, these crossbars are difficult to manufacture.

On the other hand, the blind scrubbing technique does not require expensive checks for error detection. Instead, it blindly overwrites the specified memory region at a pre-specified frequency. It has previously been implemented in FPGA devices to mitigate single-event upset (SEU) errors [137].

In general, blind scrubbing requires the storage of information about the scrub region. The memory scrubbing proposed in this thesis is also a blind scrubbing technique. However, our focus is on keeping the scrubbing cost (memory, latency per scrub operation, and power consumption) to a minimum possible while providing acceptable performance.

3.3.4. Zero Overhead ECC

There are couple of works that aim to reduce overhead of parity storage of ECC.

The work in [138, 139] proposed the method based on weight nulling, which detects errors based on even or odd parity. The LSB bit of q-bit quantized weight is used as the parity. An error is detected when an odd number of bit errors occur for a weight, and this erroneous weight is replaced with a zero value. However, this corrective action results in information loss since the erroneous weights are replaced with zeros, which may reduce the NN performance. This approach focuses primarily on error detection only and lacks explicit error correction.

Similarly, the work in [140] relies on redundant bits for 8-bit quantized weight. A modified training is proposed to ensure weight distribution in a specific range. This allows a second higher-order bit (b_6) of each weight to store the parity bits of the SEC-DED ECC. However, this method lacks multi-bit error correction capability and is not suitable for other types of ECCs. Moreover, this approach demands a large number of modified training iterations to achieve the specific weight distribution, potentially resulting in significant computational overhead. These challenges become even more pronounced when addressing scenarios involving multi-bit error correction.

The authors in [141] presented a value-aware parity insertion ECC method that relies on a specific symmetric weight distribution and the sign-magnitude representation of weights. They employed double error correction coding per 64-bit weight. For weights $|w| < 0.5$, higher bits ($b_6 b_5$), and conversely, for weights $|w| >= 0.5$, the lower bits $b_1 b_0$ are guaranteed to be 0, making them suitable for parity storage. The original weight value can be restored by overwriting the parity bits with 0. However, it is not always possible to recover the original weight because these bits are not guaranteed to be 0, when the weight distribution is more spread out, leading to a degradation in accuracy of up to ~4%. This limitation further constrains the approach from achieving zero memory overhead for ECCs requiring a larger number of parity bits. Furthermore, the approach needs extra memory overhead to identify the parity location during the decoding, and can only make corrections before the inference, as they do parity masking to recover the original weights before the inference starts.

3.3.5. Self-Healing Bayesian Neural Networks

In Section 3.1, we have discussed several works on implementing Bayesian NNs to memristor-based CiM architectures or to other AI architectures. The primary focus of those works has been largely on uncertainty estimation rather than improving *inherent fault tolerance* against non-idealities in memristor-based CiM architectures. On the other hand, several other studies focused on improving the reliability of memristor-based CiM accelerators [142, 143, 144]. Those works utilize the Bayesian framework, exploit the NVM device variation model during training to achieve resilience to device variation, or perform neural architecture search to find a robust network. However, the uncertainty estimation aspect is overlooked. Furthermore, a separate neural architecture search may potentially be a necessity in the case of other non-idealities or NVM technology. This, in turn, can result in a significantly computationally intensive approach.

This thesis proposed a self-immune Bayesian neural network [145] that is fault tolerant without compromising the quality of uncertainty estimates.

3.3.6. Online/offline Training and Re-training

In the literature, several online and offline training approaches have been proposed to improve reliability. In addition, some work proposes to retrain to improve accuracy. That is, the main objective is to improve the accuracy close to the baseline.

Online training-based methods such as [146] perform training and inference on hardware, thus accounting for hardware non-idealities. Another group of works proposes to map the trained model to edge AI accelerators and perform partial optimizations on the chip [147, 148] before deployment. However, this approach is not very practical as every neural network would have to be trained on each individual chip before deployment. Additionally, their focus is solely on mitigating the effects of post-mapping faults and variations, rather than addressing online fault tolerance. On the contrary, offline training-based methods model hardware non-idealities and incorporate them during training, e.g., via fault injection in weights or activations [30, 149]. Lastly, online retraining-based approaches such as [150] identify hardware faults, after which retraining, and weight remapping are performed. Re-training of NN is a very costly operation. Therefore, some works focused on reducing the cost of re-training via alternate layer re-training [151] and partial re-training methods [152].

In general, these works are suitable for back-propagation-enabled AI accelerator including in CiM architectures, which require additional circuitry. Nevertheless, each NN must be trained on individual chips, whereas self-healing approaches proposed in this thesis avoid fault localization, costly re-training, and weight remapping. Furthermore, explicit non-idealities injection-based methods are impractical, as they may not be generalizable across memristor technologies and chips. The reliability improvement approaches proposed in this thesis are suitable AI accelerator architectures for inference.

Part I.

Methods for Resource Scalable Predictive Uncertainty Estimation

To reiterate, in this thesis, we explore Monte Carlo Dropout, variational inference, and ensemble-based approaches for uncertainty estimation in spintronics memory-based CiM architectures.

The work was done in collaboration with colleagues from the Grenoble Institute of Technology (Grenoble INP), French Alternative Energies and Atomic Energy Commission (CEA), and Spintec Laboratory, France, as the technology partner. Specifically, circuit design, hardware evaluation, and specific module design, such as Spintronics-based Dropout modules and multi-bit memory cell design, are contributed by our colleagues.

The general target is a) perform algorithm-hardware co-design for resource and energy efficient implementation, b) reduce the number of stochastic modules to the ideal case which is one per model, c) reduce resource utilization to close to the ideal case which is a single model NNs, d) detect OOD data, and e) estimate uncertainty in prediction.

Overall, the methods presented for the first time the implementation of the In-Memory Bayesian *Binary* Neural Network with Spintronics devices. For this purpose, several novel solutions are explored. Specifically, in MC-Dropout implementation, the two modes of operation of the MTJ are explored, respectively, the stochastic feature is used to implement in the Dropout module and the deterministic behavior for the synapse storing in the crossbar array. This approach has the benefit of not requiring changes in the bit-cell of the crossbar architecture, all design changes occur in the peripheral circuits. Furthermore, since Spintronics devices operate at lower voltages (around 1V), they result in less power consumption. On the contrary, the concept of Bayesian subset parameter inference is introduced that treats only the smallest subset parameters of a NN as Bayesian. We utilize Spintronics memory as they are highly attractive for BayNN implementation due to their nano-second latency, high endurance (10^12 cycles), and low switching energy (10fJ).

4. Monte-Carlo Dropout-Based Bayesian NNs

Using Dropout for Bayesian approximation provides a practical and computationally efficient way to estimate model uncertainty. Unlike traditional dropout, which is typically used as a regularization method during training to prevent overfitting [54], Monte Carlo Dropout (MC-Dropout) extends this concept to the inference phase, as shown in Fig. 2.1.

In terms of hardware overhead, MC-Dropout typically has lower memory consumption compared to other approximation methods, as it has the same number of parameters as a conventional NN. However, it has its own challenges. In the next sections, we present the dropout-based BayNN approaches considered in this thesis.

In this chapter, we consider binary neural networks in all our work to efficiently quantify uncertainty and overcome the limited stable states of spintronics devices. Binary NNs [44] proposes a reduction of the bit precision of weights and activations with the Sign(x) function that can be described as:

$$\text{Sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{otherwise.} \end{cases}$$

4.1. Dropout-Based Bayesian Binary Neural Network

In this section, we consider Dropout applied element-wise to each neuron. However, implementing a Dropout-based Bayesian binary neural network in the CiM architecture presents algorithmic and hardware challenges. We perform full-stack optimization to overcome the challenges. The work is based on the IEEE JETCAS and ACM NanoArch [84, 85] papers.

4.1.1. Methodology

4.1.1.1. Algorithmic Description and Optimization of BayBNNs

To reiterate, in the training phase of binary NNs, the real-valued weights and activations are binarized in each forward pass using Sign(x). Therefore, the overall learning objective of an NN $\mathcal{L}$ is computed based on binary weights and activations. A regularized objective function for a dataset with a set of Q input and target output pairs, $\{(x_1, y_1), \ldots, (x_Q, y_Q)\}$, can be described as:

$$\mathcal{L} = \frac{1}{Q} \sum_{q=0}^{Q} E(y_q, \hat{y_q}) + \lambda \sum_{i=1}^{L} (|| \mathbf{W}_i ||_2^2 + || \mathbf{b}_i ||_2^2). \tag{4.1}$$

Here, the L_2 (ridge regression) regularization term (2nd part) complements the task-specific loss function $E(\cdot, \cdot)$. The hyperparameter λ controls the strength of regularization. The L_2 regularization term is minimized as the weights approach zero, but applying it to the binary weights (+1 or −1) results in a constant penalty term that cannot be optimized. Alternatively, the L_2 regularization can be applied to the real-valued proxy weights but can have no effect unless it changes the sign of the weights. For example, when the algorithm proposed by [44] is augmented with L_2 regularization, the training, loss, and error rate

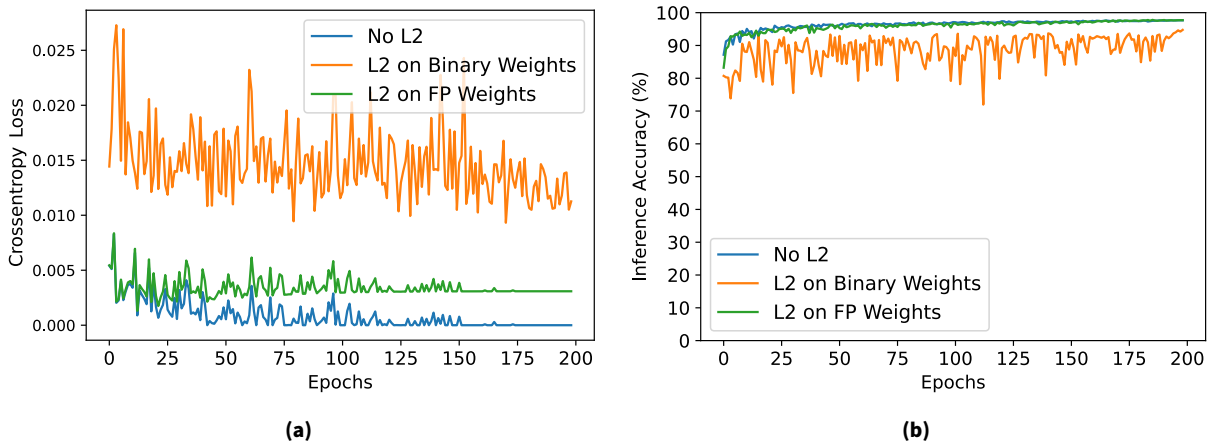


Figure 4.1.: Training curve for L_2 regularized MLP compared to un-regularized MLP on MNIST dataset. **It is recommended to view this figure in color.**

are not affected, as shown in Figure 4.1. Since adding L_2 only affects the numerical value of the weights by encouraging them to become closer to zero, the sign of the weight is usually not affected. Therefore, the dropout-based Bayesian approximation cannot be applied directly to those kinds of BNN training algorithms.

Alternatively, since the weights of the BNN are either +1 or −1, a regularization term for binary NN training should encourage the weights to be around those values. Many such regularization functions can be designed for BNN, for example, $R_1 = (\alpha - |W|)^2$, which has two minimums at $\pm\alpha$ [153]. Where, α can be a scalar value, e.g., 0.5.

In most BNN algorithms, a learnable scale factor is introduced to reduce quantization error and improve performance [45, 153]. In that case, $\alpha \in \mathbb{R}^{[C_{out},1,1,1]}$ can be incorporated into the regularization term. This will encourage the weights to be around the scale parameter. Therefore, the overall learning objective becomes:

$$\mathcal{L}_{BayBNN} = \frac{1}{Q}\sum_{1=0}^{Q} E(y_q, \hat{y_q}) + \lambda \sum_{l=1}^{L} (\alpha - |\mathbf{W}_l|)^2 + (\alpha - |\mathbf{b}_1|)^2. \tag{4.2}$$

It can be rewritten according to Equation 2.10 for Dropout-based Bayesian approximation, as:

$$\mathcal{L}_{BayBNN} = \frac{1}{Q}\sum_{q=0}^{Q} E(y_q, \hat{y_q}) + \sum_{l=1}^{L} \frac{p_l \ell^2}{2\tau Q} (\alpha - |\mathbf{W}_l|)^2. \tag{4.3}$$

The bias term can be removed for simplicity. Furthermore, the R_1 regularization term can be optimized as follows:

$$\lambda \sum (\alpha - |\mathbf{W}_l|)^2 = \lambda \sum (\alpha^2 - 2\alpha|W| + |W|^2) \tag{4.4}$$

For a large NN model, the third term of Equation 4.4 will numerically dominate the first term. Therefore, $\alpha^2 + |W|^2 \approx |W|^2$ and Equation 4.4 can be further approximated as:

$$\begin{aligned} &\approx \lambda \sum (|W|^2 - 2\alpha|W|) \\ &\approx \lambda \sum |W|(|W| - 2\alpha) \end{aligned} \tag{4.5}$$

Let, $\varpi = |W| - 2\alpha$ and substitute it with the overall objective BNN gives:

$$\mathcal{L}_{BayBNN} \approx \frac{1}{Q}\sum_{q=0}^{Q} E(y_q, \hat{y_q}) + \sum_{l=1}^{L} \frac{p_l \varpi \ell^2}{2\tau Q} (|\mathbf{W}_l|), \tag{4.6}$$

and it is equivalent to the learning objective of BayNN (Equation 2.10). Specifically, when the scale factor α is very small, i.e., close to zero, Equation 4.6 becomes exactly equal to Equation 2.10Therefore, minimizing the R_1 regularization term is equivalent to minimizing the KL divergence. As a result, a BNN trained with Dropout and R_1 is also a Bayesian approximation (BayBNN). This demonstration encouraged us to look forward to the hardware implementation of the concept.

4.1.1.2. Spintronic-based Dropout Model Description

This section describes the Spintronic-based Dropout (called in the following SpinDrop) neural network model. As mentioned in Section 2.4, in a normal Dropout, a random Bernoulli mask is sampled with a fixed dropout probability $p = [0, 1]$. As the dropout is usually implemented at the software level or in a digital fashion, the dropout probability itself is deterministic.

On the other hand, implementing the SpinDrop module with stochastic and analog components will result in a dropout probability p, which can vary due to manufacturing variations, and external environmental factors such as temperature and voltage fluctuations. Therefore, it can be considered stochastic.

With SpinDrop, the feed-forward operation of a layer l becomes:

$$\begin{aligned} \hat{p}_l &= p_l + \epsilon, \epsilon \sim \mathcal{N}(\mu, \sigma^2) \\ \hat{\mathcal{M}}_l &\sim Bernoulli(1 - \hat{p}_l) \\ \mathbf{z}^b_{l-1} &= \text{Sign}(\mathbf{z}_{l-1}) \\ \hat{\mathbf{z}}^b_{l-1} &= \mathbf{z}^b_{l-1} * \hat{\mathcal{M}}_l \\ \mathbf{W}^b_l &= \alpha * \text{Sign}(\mathbf{W}_l) \\ \mathbf{z}_l &= \phi_l(\mathbf{W}^b_l \hat{\mathbf{z}}^b_{l-1} + \mathbf{b}_l) \end{aligned} \tag{4.7}$$

Here, the mean μ and standard deviation σ come from device technology, $\hat{\mathcal{M}}$ denotes dropout mask with process variation, $*$ denotes elementwise product, $\hat{p}$ is the dropout probability with variations, ϕ is the non-linear activation function and the superscript b denotes binary matrix or vector. We show later that using the SpinDrop during training can sometimes outperform regular Dropout-based Bayesian inference.

4.1.1.3. SpinDrop Implementation

The intrinsic MTJ stochasticity due to magnetization fluctuations has a strong impact on the MRAM memory writing time. In standard MRAM memory applications, this effect is not desired.

This study implements a binary crossbar array with STT-MRAM technology to specifically allow parallel reading during inference.This allows efficiently performing one-step in-memory computing operations, which in turn is used to accelerate our proposed Bayesian binary NNs. In this array, the stochasticity of MTJs will be used in the periphery of the crossbar to allow random dropout mask generation. Thus, one or more rows of the crossbar are dropped at a time, while the MTJs used for weight storage and in-memory computation are operated in a deterministic way. We then propose to combine the stochastic and deterministic aspects of the STT-MRAM to implement the Dropout-based BayNNs approach, as explained in the previous section.

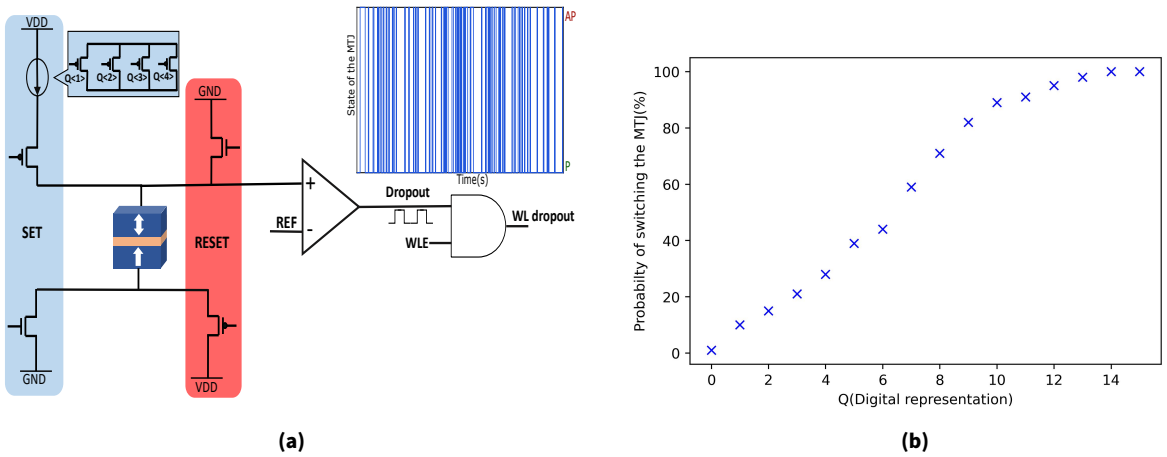


Figure 4.2.: a) Spindrop module schematic and b) the generated probability as a function of the word Q.

4.1.1.4. Magnetic Tunnel Junction as a Tuneable Bitstream for Dropout Generation

To allow the control of the current and thus of the resulting probability, a current generator is used that is driven by a digital value Q. In this case, it is obtained by five PMOS transistors connected in parallel. The first four transistors are controlled in the digital value $Q = Q_4Q_3Q_2Q_1$ (Figure 4.2), allowing a linear increase in current and the resulting probability. The fifth one (not shown) is grounded and ensures a minimum current flowing through the structure. The number of transistors can be extended or reduced depending on the chosen digital resolution. Nevertheless, it must be able to cover the entire probability range of the STT-MRAM. A bitstream with a given probability is generated through SET and RESET several alternate operations. After a "SET" write operation, the state of the MTJ is read using a sense amplifier [154] to check if the switching has occurred. The read result is the Dropout signal. Further to the reading operation, the MTJ is "RESET" to the P state to anticipate the next generation of Dropout signals.

Figure 4.2 (b) shows the evolution of the probability of MTJ switching according to the value of Q. We distinguish two regions: the first is the linear region corresponding to the transistors in the sub-threshold regime, and the second is the plateau zone that corresponds to the saturated region for the transistors.

Both MTJs and CMOS transistors from the peripheral circuits could be subject to manufacturing variations that can impact the quality of the generated bitstream, thus introducing a deviation from the target probability. To mitigate manufacturing issues, some studies proposed the use of multiple (N) MTJs [155], since the random sequences generated by several MTJs have a lower standard deviation (divided by N). To compensate for the probability deviation, another solution consists of adding a feedback loop to increase the accuracy of the generated probability [156, 157]. However, note that in our particular case, the probability deviation related to manufacturing defects is not necessarily a concern. In fact, instead of trying to target an accurate probability, these variations could be useful within Bayesian Neural Networks.

4.1.1.5. Architecture Design

To perform bitwise Bayesian inference computation within the memory, the typical classic STT-MRAM crossbar architecture needs to be modified.

In classic BNNs, the conventional matrix-vector multiplications are reduced to XNOR operations for higher efficiency. It is thus necessary to design the spintronic-based bit-cell that allows performing this particular operation. Several digital [158] and analog implementations have been presented in the literature. Also, analog solutions could be used, that take advantage of the summation of currents according to Kirchhoff's law.

The proposed architecture is organized around the crossbar MTJ array, in which each trained weight is stored in a unit represented by two 1T-1MTJ cells, as in conventional BNN crossbars. One SpinDrop module, described in the previous section, is connected to each wordline pair to implement the Dropout concept described earlier. A bitline conditioning circuit is used to set the bitline according to the inputs. An ADC per sensed bitline and a sourceline conditioning circuit are connected to the output, to sense and convert the state of the enabled cells. In addition to that, they are provided with a CMOS-based Accumulator-Adder module to sum up the partial Matrix-Vector product results according to Equation 2.2. Finally, a comparator and a running average CMOS circuit complete the schematic, ensuring that the computation achieves the predictive mean needed for the Bayesian inference. The sourceline periphery (ADC, accumulator-adder, and so on) can be shared by multiple sourcelines using MUXes to save chip area by reusing temporary un-useful components. The output of the CMOS periphery of the crossbar can then be provided to input to the next crossbar that also respects the same architectural design.

Moreover, as Bayesian inference with Dropout approximation requires spatial and temporal independence, the probability that a neuron is dropped is independent of one another and also from one input to another. To achieve temporal independence, the proposed SpinDrop module randomly activates and inactivates each word-line with probability p and $1-p$, respectively, during each forward path. In addition, to achieve spatial independence, a separate SpinDrop module is used for each word-line of the crossbar. Figure 4.4 (b) describes the Bayesian inference systematically for a linear layer with 10 Monte Carlo samples (T=10).

As mentioned before, the presented architecture produces a weighted sum calculation of a single layer. The output of a layer is then used as the input to the next layer. Each input neuron is mapped to two rows, and each output neuron is mapped to one column, that is to say, two word-lines feed one input neuron while a column line feeds one output neuron. Each pair of cells thus represent the connection between an input neuron and an output neuron. To evaluate the output of a neuron, the following steps have to be performed serially for each layer of the BayBNN.

At the beginning of the inference operation, each of the SpinDrop modules randomly drops its corresponding neuron with the Dropout probability, which is implemented by enabling or disabling the respective wordlines. As only a limited number of cells per bitline can be reliably sensed at once [159], our architecture supports the computation of group-wise CiM operations. Based on parallel measurable cells $\mathbf{z}^{\mathrm{CiM}}$, the array with $\mathbf{z}_{l-1}$ inputs (stored in $2 \times \mathbf{z}_{l-1}$ rows) is split into groups of $\mathbf{z}^{\mathrm{groups}} = \mathbf{z}_{l-1}/\mathbf{z}^{\mathrm{CiM}}$. Each group is selected one by one via the wordline decoder, and SpinDrop modules are used to dropout an input by disabling the wordline of the respective input neuron. The implementation of the SpinDrop module is performed by adding a pass-gate that allows access either to the classical decoder or to the stochastic wordline (WL). Here, one SpinDrop module is used per wordline, but this can be reduced down to four (depending on the maximal CiM operation group size) and thus multiplexed among the different group operations. The ADC is used to calculate the result of the XNOR operations, which is then summed up in the accumulator-adder module. After all the groups have been summed up, the comparator performs a threshold function. The threshold function chosen here is the $\mathrm{Sign}(x)$ function, which is used to binarize the weights and the activation in a typical BNN. The MUXes can be used to map multiple layers in the same crossbar and evaluate them one by one.

To get the predictive performance of a Dropout based BayNN, the average result of all individual Monte-Carlo inferences with SpinDrop enabled has to be calculated. The results of the neurons of the output layer are used in a running average to evaluate the predictive performance and uncertainty of BayNN. Therefore, the calculated mean value is the Bayesian Inference result, and variation is the corresponding confidence (uncertainty) in this result.

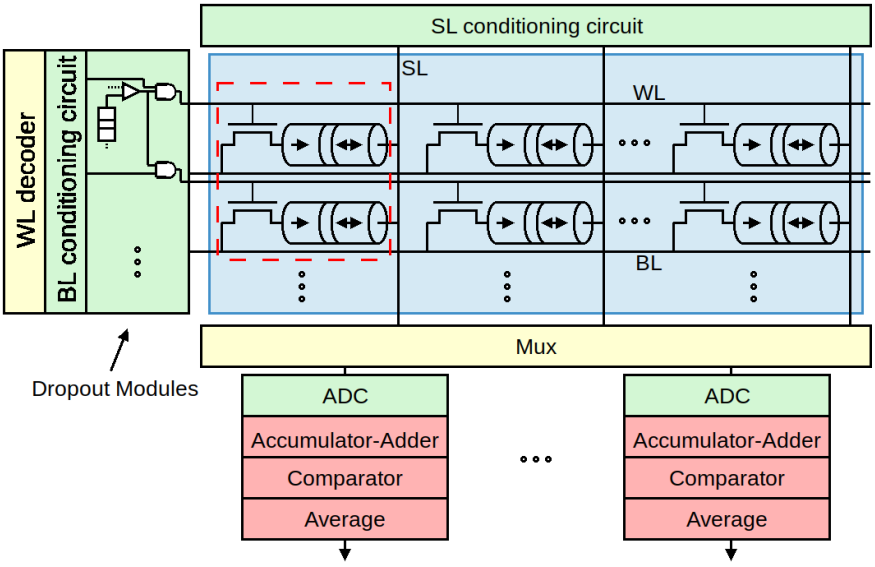


Figure 4.3.: Inference architecture for BayBNN inference

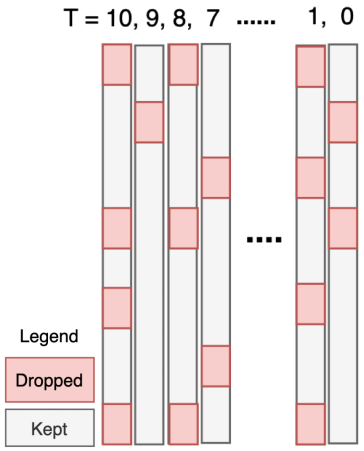


Figure 4.4.: Depicting temporal independence in Dropped neurons with proposed SpinDrop. **It is recommended to view this figure in color.**

4.1.2. Evaluation

4.1.2.1. Evaluation Setup

For the architectural simulation, we first obtained the circuit characteristics of the peripheral blocks described in Section 4.1.1.3. The Acumulator-Adder, Comparator, and Averaging circuits were synthesized with the Synopsys Design Compiler using the TSMC 40 nm low-power PDK-based standard IP libraries. The decoding and sensing for the CiM operation were evaluated in the circuit-level array using NVsim [160]. To do that, we adjusted the NVsim simulation with the data presented in Section 4.1.2.2 to account for four active cells, thus modeling the CiM operation. We have also replaced the single-bit sense amplifiers with multi-bit ADCs. The results for each individual component are shown in Table 4.1.

To evaluate both predictive performance and uncertainty estimations, we trained a) an MLP with four layers on the MNIST dataset, b) a LeNet-5 topology on the MNIST dataset, c) a VGG topology (nine layers) on the CIFAR-10 dataset, and d) ResNet-18 (eighteen layers) on the CIFAR-10 and CIFAR-100 datasets. Specifically, the MLP has 256 neurons per layer for three hidden layers, and the last one being dependent on the dataset has 10 neurons.

For training the models, we used the Adam optimization algorithm with default settings in PyTorch to minimize the cross-entropy loss function with $\lambda = 1 \times 10^{-5}$. The model precision τ can be derived from the value of λ. We trained the MNIST dataset for longer (400 epochs) than the CIFAR-10 and CIFAR-100 datasets (100 epochs) due to the network sizes. We have applied RandomHorizontalFlip and RandomResizedCrop type data augmentation to CIFAR-10 datasets to improve accuracy. However, no data augmentation is applied to the MNIST dataset.

Moreover, to show the scalability of our proposed approach to even larger topologies and harder tasks, several real-world biomedical image segmentation and classification datasets are evaluated on state-of-the-art topologies. For biomedical image classification, we have trained DenseNet-121 (121 layers) topology on pneumonia detection (from the chest X-Rays) dataset.

On the other hand, for biomedical image segmentation, Digital Retinal Images for Vessel Extraction (DRIVE) [161], breast ultrasound scans (for breast cancer) [162], and Mitochondrial Electron Microscopy [163] datasets are evaluated on U-Net [164], Bayesian SegNet [165], and Feature Pyramid Network (FPN) [166] with ResNet-50 (50 layers) as feature extractor, respectively. The DRIVE dataset comprises 40 images with a size of 584 by 565 pixels, 20 of which are used for training and the other 20 for testing. The Breast Ultrasound Scan dataset (breast cancer) is used for the early detection of breast cancer, which is one of the most common causes of death among women worldwide. The dataset is categorized into three classes: normal, benign, and malignant images. There are a total of 780 images, with an average image size of 500 by 500 pixels. The Electron Microscopy Dataset depicts a $5 \times 5 \times 5nm$ section of the CA1 hippocampal area of the brain, which corresponds to a $1065 \times 2048 \times 1536$ volume. Since the size of each image is too large to fit in NN topologies, we have patched each image and its corresponding mask into 256×256 masks for training.

The metrics used to determine the performance of segmentation tasks are pixel-wise accuracy, Intersection-Over-Union (IoU), Sensitivity, Specificity, Area Under the ROC Curve (AUC), F1 Score, and precision. IoU is one of the most commonly used metrics in segmentation tasks and is the ratio of the area of overlap and the area of union. The pixel-wise accuracy states the percentage of pixels in the predicted image that are classified correctly. The specificity represents the proportion of actual negative cases accurately recognized by the model. Sensitivity represents the proportion of actual positive cases correctly identified by the model. AUC summarizes the true positive and false positive rates into a single number, with a higher value indicating better performance. The F1 score measures the accuracy of a model in a dataset by integrating precision and recalls into one metric. Lastly, precision is the proportion of positive predictions that are actually correct.

To show the application of BayBNN in detecting out-of-distribution data and noisy input (aleatoric uncertainty), we have used the validation subset Fashion-MNIST dataset and randomly generated five datasets:

1. Inverted MNIST (Dataset D_1). Each pixel of the input image is inverted, i.e., dark becomes white,
2. Unit Gaussian noise $\mathcal{N}(0, 1)$ (Dataset D_2). Each pixel of the input image is Gaussian noise,
3. Continuous uniformly distributed noise (Dataset D_3). Similar to Gaussian noise, each pixel of the input image is uniform noise,
4. Randomly rotated (±90) MNIST images (Dataset D_4),
5. MNIST with random Gaussian noisy background (Dataset D_5), and
6. Fashion-MNIST dataset (Dataset D_6).

In total, 10000 input data for each dataset were used and each of the data has the same shape (28×28) as needed for the MNIST dataset. The random noise in each dataset changes for each Monte-Carlo sample for Bayesian NN.

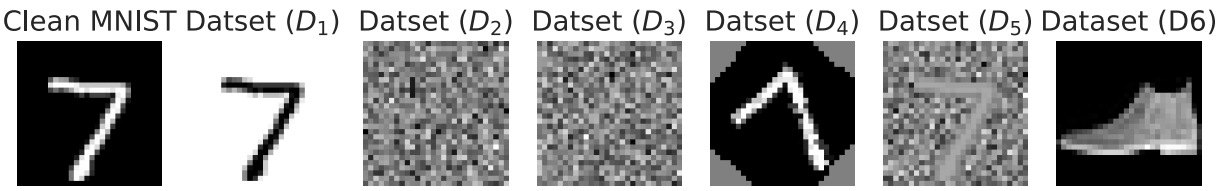


Figure 4.5.: Datasets to measure the detection capability of out-of-distribution data. **It is recommended to view this figure in color.**

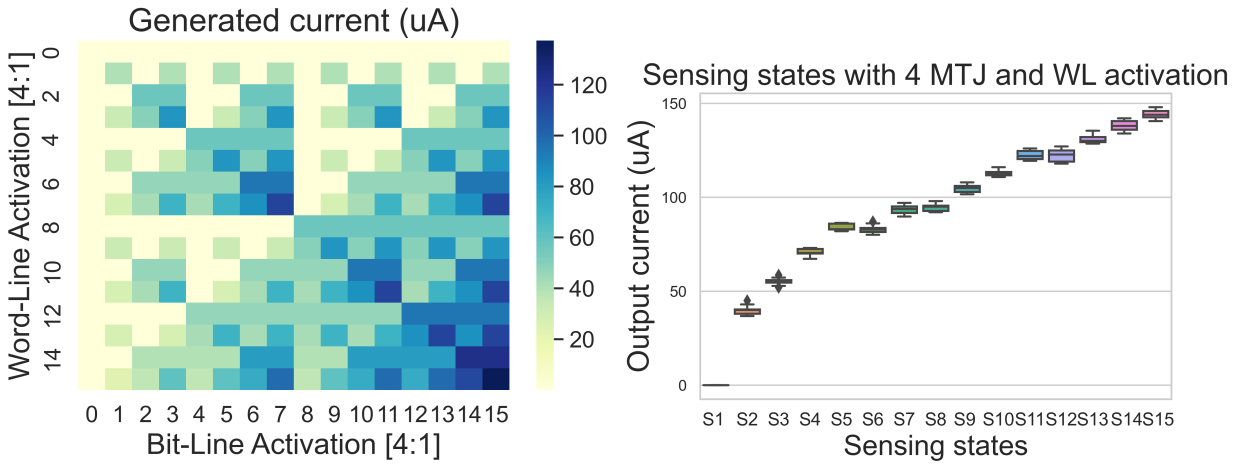


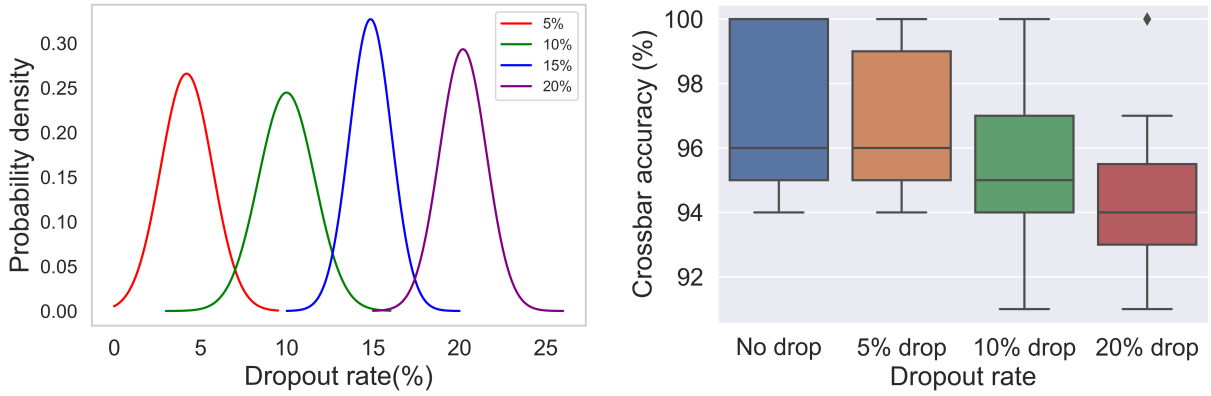
Figure 4.6.: Evaluation of 4 bit-cells. **It is recommended to view this figure in color.**

To evaluate the robustness to thermal fluctuation (epistemic uncertainty) of the proposed SpinDrop BayBNN, we inject random Gaussian noise $\mathcal{N}(0, I)$ into the weighted sum of each layer. Furthermore, we have explored different implementations of the SpinDrop in terms of dropout probability and location of the Dropout layer to check their impact on inference accuracy. For our hardware analysis, the SpinDrop module was used in all layers and crossbars.

4.1.2.2. Cell-level Simulation

To evaluate and understand the global effects of stochasticity at the bit cell level, an initial study of the crossbar output current variability, taking into account various MTJ states, was conducted. An example of which is shown on the map in Figure 4.6. Our objective was to determine the range of operation (best-case; worst-case) for the currents and the impact of the MTJ states on the output current. Moreover, this study will also allow us to determine the maximum size of the crossbar that is still functional for the proposed SpinDrop BayBNN architecture. The simulation has been performed through several Monte Carlo samples (i.e., 30) on the Bit-cell, considering both MTJ and (selecting) CMOS device variations. We extracted 15 different crossbar sensing states, as shown in Figure 4.6. We notice that when the current is higher, that is, when the number of MTJ active in the crossbar is important, the deviation of the output current related to device variability also increases. Our evaluation shows that the same values of currents are obtained for different bit-line and word-line activations. We also observe a peak current of 140μA. These current levels will also be used for power consumption estimates in Section 4.1.2.4.

For the SpinDrop module, a second evaluation was performed. In order to control the switching probabilities of the MTJs, the current flowing through it is adjusted thanks to the value of Q, as shown in Section 4.1.1.4. The current varies from 80 to 150 µA for a duration of 10 ns for the SET signal. For the RESET signal, an amplitude of 300 µA for a duration of 4 ns was used. This current has to be high enough to ensure that the MTJ is reliably switched.



(a) Impact of process variations on the Dropout rate (SpinDrop module).

(b) Impact of Dropout on accuracy (crossbar).

Figure 4.7.: Dropout non-idealities. **It is recommended to view this figure in color.**

The evaluation of the SpinDrop module aims to assess how accurately it generates the dropout mask with the predefined probability. This allows us to quantify the effects of variability and stochasticity on dropout probability.

To do this, we performed several Monte Carlo analyses (i.e., 100 switchings of the MTJ within 20 Monte Carlo runs, which is equivalent to 2000 Monte Carlo simulations on an MTJ). Note that the Monte Carlo simulations performed here are different from the Monte Carlo sampling required for Bayesian inference. The probability of dropping the neuron from the crossbar output will be equal to:

$$P_{Dropout} = 1 - P_{Switching} \tag{4.8}$$

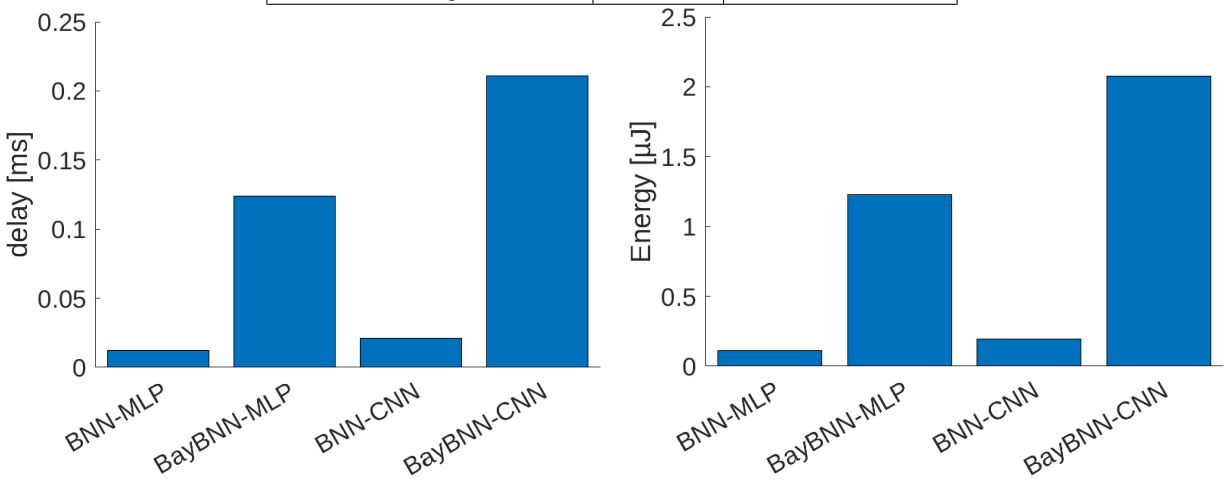
As we can see in Figure 4.7, the generated probability is not fixed but follows a Gaussian distribution. In fact, when the probability of switching the MTJ approaches 100%, it becomes difficult to generate an accurate probability because we are limited by the saturation regime of the transistors (as explained in Section 4.1.1.4). As we can see in Figure 4.2, the SpinDrop module is no longer linear and the values are quite close to each other. These cell-level evaluation results will be used in circuit-level simulation.

4.1.2.3. Circuit-level simulation

A 20×10 crossbar array is implemented according to the architecture presented in Figure 4.3. With this implementation, we intend to illustrate the Dropout concept impact on a larger crossbar with accurate Spice-level simulation. For this validation, the crossbar outputs are connected to a winner-takes-all (WTA) circuit [167]. WTA circuits inhibit all its inputs except the one corresponding to the highest current at the output of the crossbar. Therefore, the ArgMax prediction is directly implemented in the crossbar. To implement the proposed SpinDrop module, another peripheral circuit is added that selectively allows the activation of the previous layer to stochastically activate the wordline of the current layer. In Figure 4.7, the impact of the Dropout rate on the overall accuracy is depicted. Firstly, we noticed that the overall accuracy is slightly affected. In fact, when a wordline is dropped, all output currents are impacted in the same way, and the output with the highest current will still be declared the winner by the WTA circuit. However, dropout certainly has an impact on accuracy, especially around 20%. Starting with this rate, the probability of dropping all the word-lines of the crossbar is increased, leading to potential classification errors. In addition, when coupling with the Monte Carlo process variation simulations, the loss of accuracy could be even higher.

Table 4.1.: Circuit-level characteristics

circuit	delay [ns]	dynamic energy [pJ]
Memory (+Decoding/Sensing)	1.87	42.839
Accumulator-adder	2.77	0.0621
Comparator	2.87	0.0106
Average	22.15	18.4243



(a) Total Delay

(b) Total energy consumption

Figure 4.8.: Energy and Delay for MLP and CNN inferences evaluated for BNN and SpinDrop-based BayBNN with 10 MC samples taken to achieve similar accuracy.

4.1.2.4. Architecture Simulation

Data from Table 4.1 were used to approximate the delay and inference energy of a single forward pass of an image from the MNIST dataset. Figure 4.8 shows the delay and energy consumption of both the MLP and CNN topologies, implemented with conventional binary NN and Bayesian binary NN. In each graph, we compare the BNN with the proposed SpinDrop BayBNN with 10 MC samples taken for the predictive performance and a CiM group size of $\mathbf{z}^{CiM} = 4$. Taking more MC samples does not improve predictive performance, as shown in Figure 4.12.

Note that BayNN implementation tends to be more computationally expensive in contrast with conventional BNN implementation since one must compute the expectation (average) of many inference results. To perform a single inference with the BayBNN, each layer is evaluated using multiple CiM operations to calculate the results of the matrix-vector multiplication between the input to the layer and its weights. Each of these CiM operations gives a partial sum of the final neuron output, which is generated by adding up the partial sums in the accumulator-adder module. After all partial sums are added up, the comparator is used to implement the threshold activation (Sign(.)) function. This is done for all layers to get a prediction for a single inference. During BayNN inference, this procedure has to be performed multiple times, depending on the number of Monte Carlo samples, in which the SpinDrop module is used to randomly drop the neurons during the CiM operation. To get the final result, the average circuit computes the running average of the MC runs. However, the main contributions to energy and delay are extra the individual forward passes occurring during the predictive mean computation. Therefore, the total delay and energy consumption scale linearly with the number of Monte Carlo samples for Bayesian inference.

In Table 4.2, the proposed approach is compared with SOTA implementations in terms of energy consumption, all based on the MNIST dataset. Compared to the NVM implementation in work [99], the proposed approach with STT-MRAM technology is ×4.65 less energy-consuming. It should be noted that their RRAM implementation is done only with 2 hidden layers, whereas our implementation is realized on a Lenet-5 topology. When considering the FPGA implementation, the energy consumption is ×10 times

Table 4.2.: Hardware comparison with other implementation

Related works	Implementation	Bit resolution	Energy (µJ/Image)
R.Cai et al.[168]	FPGA	8-bit	18.97
X.Jia et al.[169]	FPGA	8-bit	46.00
H.Awano et al. [95]	FPGA	7-bit	21.09
A. Malhotra et al. [99]	RRAM	4-bit	9.30
Proposed implementation	STT-MRAM	1-bit	**2.00**

lower compared to an implementation with two hidden layers in works [168, 95]. With the same Lenet-5 topology as ours, the energy evaluated is ×20 better compared to [169].

4.1.2.5. Uncertainty Estimation

The evaluation in this section is performed with a Dropout probability of 20%. In addition, the dropout is applied to all layers and crossbar arrays.

Ood Analysis and Detection In critical applications, any NN results must be correct and trustworthy, otherwise an error flag should be raised. Such kinds of NNs should only predict an answer when the input distribution matches the trained one, D_{train}. Point Estimate NNs cannot infer "Undecided" if they receive out-of-distribution data. We found that when a BNN model trained on the MNIST dataset receives out-of-distribution data, it predicts random MNIST labels. For example, it mostly predicts MNIST labels 0 (with a frequency of 28.28%) and 8 (with a frequency of 62.17%) when the dataset D_2 (random Gaussian noise) is applied as input. In fact, our evaluated point-estimate NN predicts 66.13% of the random Gaussian D_2 and 97.77% of uniform D_3 inputs with 100% confidence that the input is a handwritten digit.

If a fail-safe NN model receives in-training distribution data, it should confidently predict the correct label (that is, the prediction probability should be close to 100%) and for each of the MC samples to have low variance in the prediction probability. In contrast, the variance in prediction probability for out-of-distribution data is expected to be high. We have utilized the expected behavior of a fail-safe system to introduce two metrics to detect out-of-distribution data. Specifically, NN only predicts when it is highly confident (prediction probability $\geq$ 95%) and has low uncertainty (that means highly certain) in prediction (quantile score of 10%). A quantile score of 10% has 90% of the confidence score of all MC samples above that value, that is, low variance. Consequently, the proposed SpinDrop BayBNN can detect up to 100% of out-of-distribution data from the dataset $D_1 \cdots D_6$, as shown in Figure 4.10.

To further emphasize the significance of uncertainty in detecting out-of-distribution data, we have also assessed the situation in which the model predicts when uncertainty is moderate (or moderately certain) or high (that means not certain). Nonetheless, the prediction confidence is still considered high. A quantile score of 50% (median of the MC samples) and 90% is considered for a moderate and high uncertainty analysis, respectively. If the model predicts despite moderate uncertainty, its ability to detect out-of-distribution data could decrease by as much as 61.68%. However, the model cannot detect any out-of-distribution data for most of the datasets when it predicts despite a high uncertainty, as shown in Figure 4.10. This demonstrates the significance of uncertainty estimation in the detection of out-of-distribution data.

Additionally, our proposed method is also robust against random data poisoning. When 20% of the MNIST validation data is poisoned with random Gaussian noise (dataset D_2) with random labels, our proposed Dropout based Bayesian BNN can detect 84.45% of poisoned data and achieves an accuracy of 96.28% on predicted inputs. On the contrary, the accuracy of the point-estimate NN decreases from 98.83% to 81.19%. Consequently, our proposed SpinDrop BayBNN improves inference accuracy by 15%, in addition to obtaining fail-safe properties. Data poisoning in our context is considered to occur when 1. An adversary

Table 4.3.: Predictive performance comparison with SOTA methods on CIFAR-10 data. Results for related works reported based on [47].

Topology	Method	Bit-width (W/A)	Acc.(%)
ResNet-18	FP	32/32	93.0
	RAD [170]	1/1	90.0
	IRNet [47]	1/1	91.05
	BNN+ [153]	1/1	88.51
	DIR-Net [171]	1/1	92.8
	SpinDrop BayBNN	1/1	**90.48**
VGG Small	FP	32/32	91.7
	LAD [172]	1/1	87.7
	XNOR [45]	1/1	89.8
	BNN [44]	1/1	89.9
	RAD [170]	1/1	90.0
	IRNet [47]	1/1	90.4
	DIR-Net [171]	1/1	91.1
	SpinDrop BayBNN	1/1	**91.5**

Table 4.4.: Predictive performance comparison with SOTA methods on CIFAR-100 data.

Topology	Method	Bit-width (W/A)	Acc.(%)
ResNet-18	FP	32/32	70
	IRNet [47]	1/1	61.03
	BNN+ [153]	1/1	64.37
	SpinDrop BayBNN	1/1	**67.97**

is aware of the inference data of the model, and has the power to alter a small fraction of the inference data in order to degrade the trained model's overall accuracy, or 2. The data generation process is noisy.

We have also conducted an experiment similar to [35] with a continuously rotated image of digit 1 in the LeNet-5 topology, as shown in Figure 4.11. We performed 100 stochastic forward passes to obtain the softmax input, the output of the final fully connected layer, and the softmax output. Recall that the softmax output is the class probability based on the output of the NN. For the 12 images, the point estimate BNN predicts classes [1 1 1 0 0 5 3 3 5 2 5 7]. The figure shows, initially, that the majority of the stochastic softmax output for level one (correct level) is close to one (100% confident) and there is very low variance in the predictions. However, as the rotation increases, the softmax output for level one reduces, and other (incorrect) levels increase. Nevertheless, the point estimate model predicts, even though the uncertainty in the prediction is very high. In this scenario, it is reasonable for the model to reject the prediction and request a label from an external annotator for this input. The uncertainty of the model can be obtained from the entropy or variation across stochastic runs.

Hardware Uncertainty Due to the non-idealities of spintronic technologies, the on-chip model introduces additional in-field uncertainty. For example, dynamic thermal fluctuations cause noisy weighted sums in the

Table 4.5.: The analysis of the proposed SpinDrop BayBNN on three biomedical segmentation datasets trained on three SOTA topologies. A Dropout probability of 20% is used with 20 Monte Carlo samples for the Bayesian inference.

Topology	Dataset	Method	Bit-width (W/A)	Pixel-Acc	IoU	AUC	F1	Sensitivity	Specificity	Precision
U-Net	DRIVE	MC-Dropout	32/32	96.35%	66.35%	98.22%	80.17%	84.39%	97.50%	76.35%
		SpinDrop BayBNN	1/4	**96.49%**	**67.44%**	**98.26%**	**80.90%**	**84.96%**	**97.60%**	**77.21%**
Bayesian SegNet	Breast Cancer	MC-Dropout	32/32	97.83%	71.82%	96.40%	83.21%	78.23%	99.28%	88.86%
		SpinDrop BayBNN	1/4	**97.47%**	**68.17%**	**95.48%**	**80.13%**	**74.25%**	**99.18%**	**87.02%**
FPN (ResNet-50)	Mitochondrial EC	MC-Dropout	32/32	98.68%	77.89%	96.49%	87.29%	85.55%	99.41%	89.12%
		SpinDrop BayBNN	1/4	**98.61%**	**77.63%**	**97.19%**	**86.61%**	**84.59%**	**99.40%**	**88.72%**

Table 4.6.: Test the prediction accuracy (%) of the proposed SpinDrop BayBNN in comparison to the 32-bit MC-Dropout method. The proposed approach employs Sign(.) activation. Accuracy is reported following 20 Monte Carlo samples for Bayesian inference.

Topology	Dataset	Method	Bit-width (W/A)	Inference Accuracy
DenseNet-121	CXR Pneumonia	MC-Dropout (HTanh)	32/32	85.00%
		MC-Dropout (ReLU)	32/32	86.66%
		BNN+	1/1	88.14%
		SpinDrop BayBNN	1/1	**91.19%**

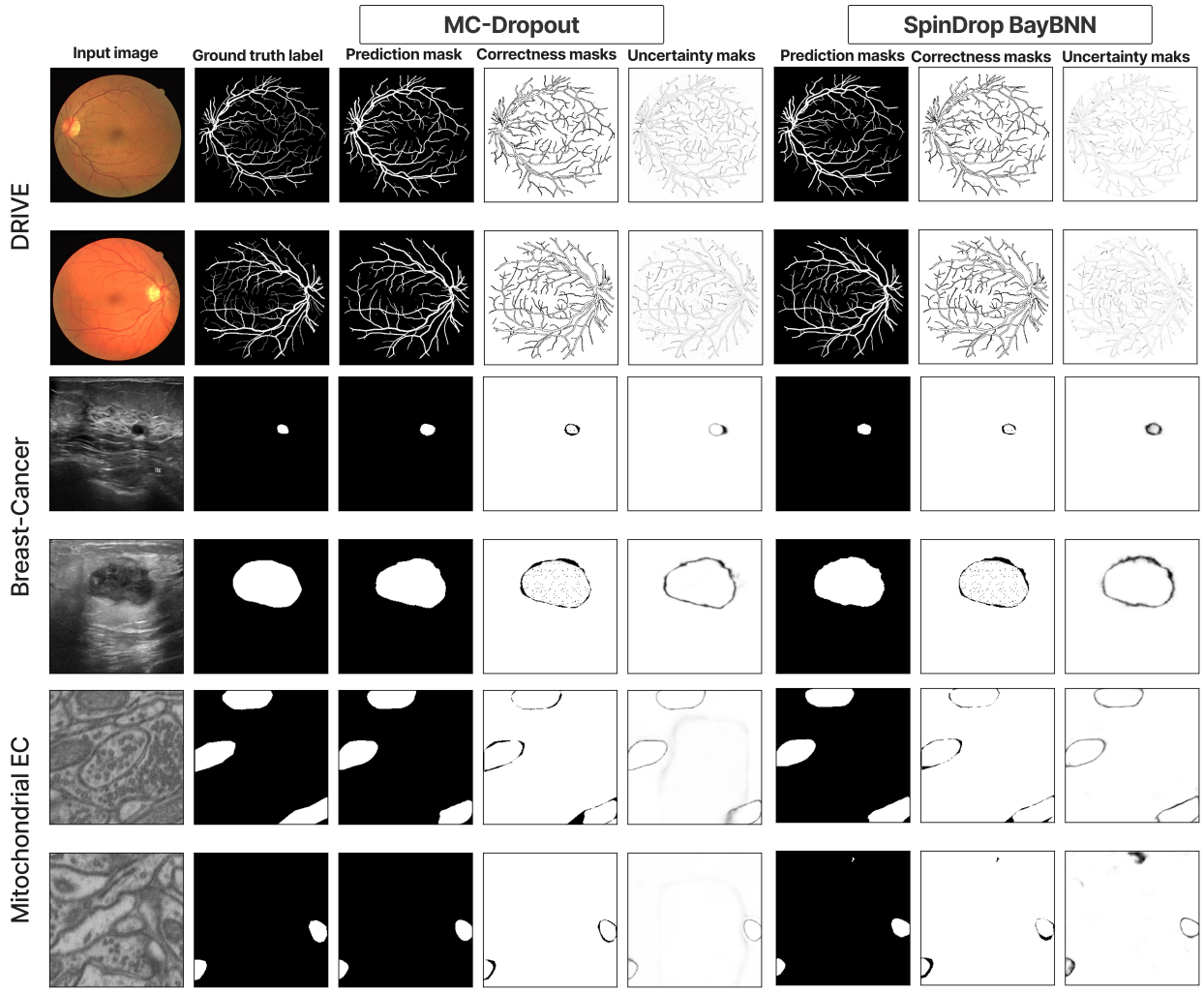


Figure 4.9.: Results of semantic segmentation and estimations of uncertainty for the DRIVE, Breast Cancer, and Mitochondrial Electron Microscopy datasets. Each row represents a single sample and contains the input image along with the ground truth, prediction, correctness, and uncertainty maps for both the MC-Dropout and SpinDrop BayBNN methods. The correctness map is a binary representation of correct and incorrect predictions. The uncertainty map is the normalized [0,1] map of uncertainty values derived after 20 Monte Carlo samples. For prediction masks, a threshold of 0.5 is applied. Correct and certain regions are displayed in white on the correctness and confidence maps, respectively. Similarly, an incorrect or uncertain region is shown in black. **It is recommended to view this figure in color.**

crossbar array. The proposed SpinDrop BayBNN can be leveraged to handle model uncertainty and attain robustness to dynamic thermal fluctuations. In our analysis of thermal fluctuations, inference accuracy of the BNN reduces to 90.16%, 76.41%, and 69.89% respectively for noise strength of $\mathcal{N}(0, I), I \in (3, 4, 5)$. However, the inference accuracy of the proposed SpinDrop BayBNN does not reduce when 50% or fewer MC-samples are noisy due to thermal fluctuations, as shown in Figure 4.12 (a).

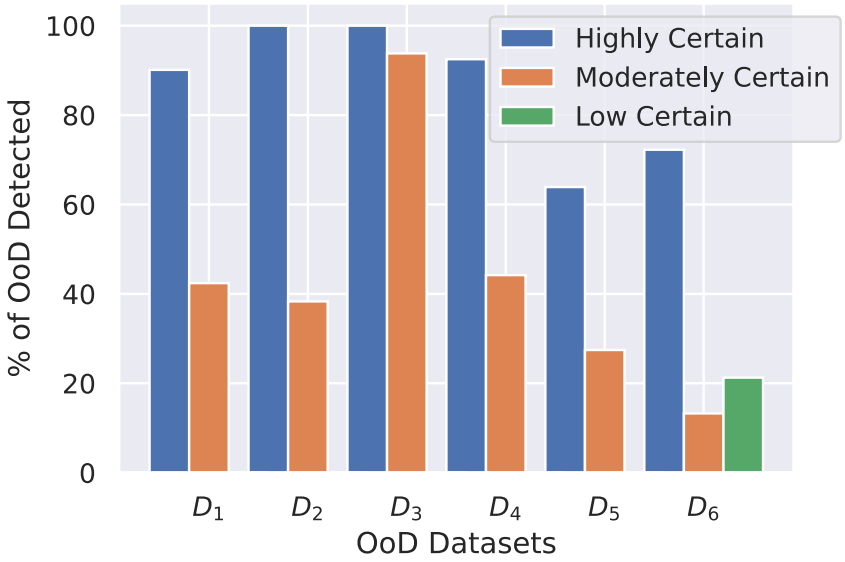


Figure 4.10.: Capability of the proposed method in detecting out-of-distribution (OOD) datasets D_1 to D_6. Evaluated on a model trained for MNIST dataset with a Dropout probability of 15%. **It is recommended to view this figure in color.**

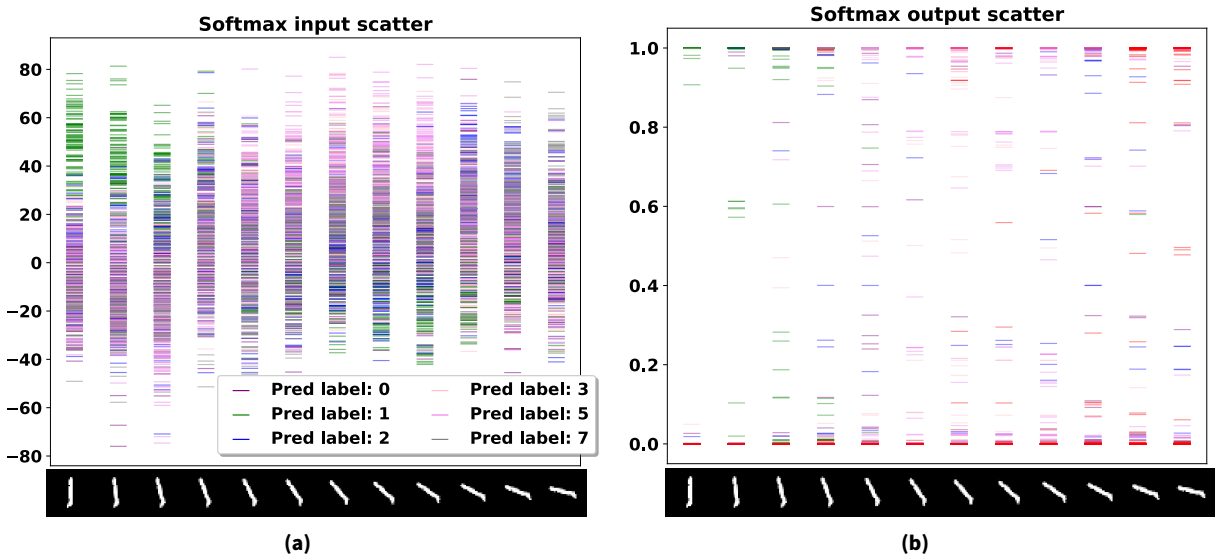


Figure 4.11.: Twelve continuously rotated images of the digit 1. Point estimate BNN classifies inputs as digits [1 1 1 0 0 5 3 3 5 2 5 7], even though model uncertainty is extremely large as the rotation is increased. **It is recommended to view this figure in color.**

Epistemic Uncertainty The model uncertainty results for the biomedical segmentation tasks are qualitatively depicted in Figure 4.9. The uncertainty masks show the pixel-wise uncertainty for each prediction. It can be observed that the proposed SpinDrop BayBNN method generates model uncertainty similar to the MC-Dropout method. Ideally, uncertainty is expected to be high around misclassified pixels and low around correctly classified pixels. Overall, it can be observed in Figure 4.9 that the uncertainty is high around the misclassified pixels, but correctly classified pixels have a low uncertainty. In general, MC-Dropout produces slightly stronger uncertainty masks as a higher Dropout probability (50%) is used.

4.1.2.6. Analysis of the Predictive Performance

Here, we thoroughly analyze the predictive performance of the proposed approach and compare it with related works.

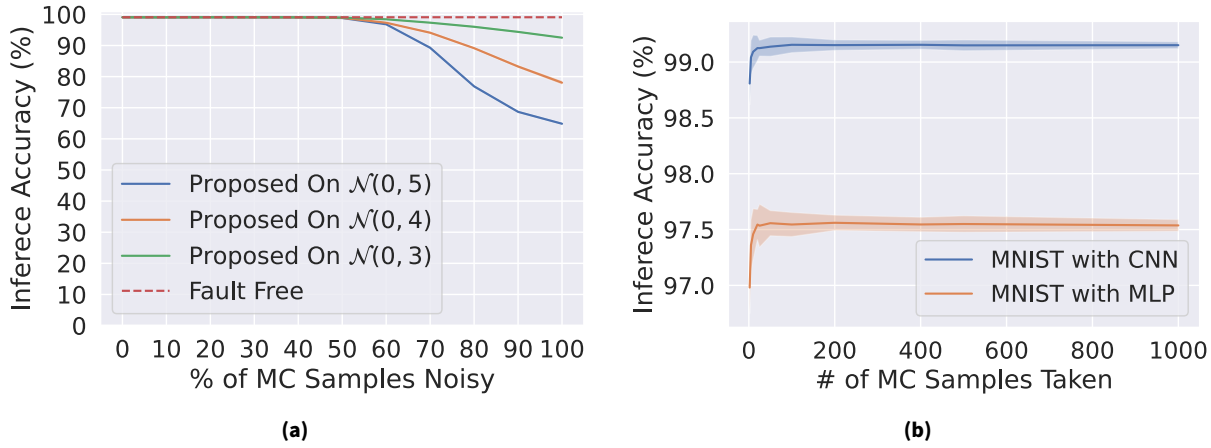


Figure 4.12.: a) Impact of thermal variations on the inference accuracy. Evaluated on MNIST dataset with 15% spintronic-Dropout, b) Impact of the number of MC samples taken on accuracy. **It is recommended to view this figure in color.**

Table 4.7.: Predictive performance of BayNN, point estimate NN (Pe. NN), and proposed SpinDrop BayBNN with spintronic-based Dropout with (w/) and without (w/o) variations in the SpinDrop module. Evaluated for MNIST dataset with different non-linearity functions.

	BayNN [35]		SpinDrop BayBNN	
Variations	w/o		w/o	w/
Activation	ReLU	HTanh	Sign	Sign
Pe. NN	97.17	97.2	99.09	99.09
T=50	99.47	99.13	99.22	99.28
T=100	99.48	99.09	99.21	99.28
T=1000	99.42	99.1	99.22	99.31

Comparison With State-of-the-Art Algorithms The predictive performance of our BayBNN, assuming that there are no variations in the SpinDrop module, is comparable to the full precision BayNN [35] as shown in Table 4.7 for the MNIST dataset on LeNet-5 CNN topology. For a fair comparison, we have used the same network size used in [35]. They use MaxPool as the activation for the convolution layers, whereas our proposed SpinDrop BayBNN uses the Sign(.) activation function for all the layers where activations are applied. When hyperbolic tangent (Htanh) is used as the activation function, our proposed SpinDrop BayBNN achieves a slight improvement in accuracy for up to 1000 MC samples. However, the predictive performance of [35] with ReLU activation is slightly better than that of the others. This analysis shows that binarizing BayNN can still achieve predictive performance comparable to full-precision BayNNs. However, the hardware implementation for our solution may lead to a smaller area and better power-performance product thanks to simpler activation functions and binary weights. As mentioned, a smaller dropout probability of 20% is used in our analysis. Additionally, we have explored many different kinds of implementations with different locations for the Dropout layer and dropout rates. They are discussed further in the following Sections.

Furthermore, our proposed BayBNN achieves an inference accuracy comparable to the SOTA point estimate BNN algorithm for VGG, and the ResNet-18 CNN topology with CIFAR-10. However, in the CIFAR-100 datasets, our method outperforms the SOTA point estimate BNN algorithm by 3.6%, as summarized in Tables 4.3 and 4.4. In the worst-case scenario, performance is only 0.57% and 2.3% worse than SOTA IR-Net [47], and DIR-Net [171], respectively, in ResNet-18 topology. In general, we have found that taking T Monte-Carlo samples and averaging them increases the inference accuracy of the NN model trained with and without Dropout.

For the biomedical image classification task, our proposed method outperforms both full precision and binarized NNs by up to 3.04%, as shown in Table 4.6. Our method is trained with a lower dropout probability (10%) compared to the MC-Dropout method. When a higher dropout probability is used, e.g., 20%, the performance of our method is reduced by 3%. Similar to other datasets, we have found that using Monte

Table 4.8.: Predictive performance of BNN and BayBNN for MNIST dataset on LeNet-5 and CIFAR-10 on VGG topology, considering with and without variations σ of the SpinDrop module.

Variations	w/o	w/			
		Inference			Trained
σ	0x	1x	2x	3x	1x
MNIST					
BNN	99.09	99.09	99.09	99.09	98.98
SpinDrop BayBNN	99.22	99.28	99.27	99.27	99.13
CIFAR-10					
BNN	88.86	88.86	88.86	88.86	88.33
SpinDrop BayBNN	89.49	89.52	89.48	89.51	89.22

Carlo sampling for Bayesian inference improves the predictive performance. We have used 20 ($T = 20$) samples for Bayesian inference.

Moreover, for biomedical image segmentation tasks, the proposed method performs similarly on all matrices to the 32-bit full-precision MC-Dropout method, with up to a 1.09% improvement in IoU score. In the worst case, our method achieves a 3.65% lower IoU score. The results are summarized in Table 4.5. Similar to other datasets, we have taken 20 ($T = 20$) samples for Bayesian inference. We used higher-precision 4-bit activation for segmentation tasks, as they are much more difficult than classification tasks. We have used the algorithm proposed in [173] for activation quantization. Since weights are still kept at 1-bit, no changes to the crossbar structure are required. Only peripheral changes, such as a higher bit-resolution ADC, are required.

Two examples of predictive performance for each dataset are shown in Figure 4.9. In addition, the performance of SpinDrop BayBNN is compared qualitatively to the full-precision MC-Dropout method. Our observations show that the proposed SpinDrop BayBNN performs similarly to MC-Dropout, that is, predicts similar segmentation masks. In general, most of the miss-classified pixels are on the boundary of ground truth masks.

Performance of BayBNN with SpinDrop Although the predictive performance of BayBNN (algorithmic, i.e., no variations in the SpinDrop module) is comparable to the full precision implementation, it should also be robust to the variations of the Spintronic-based implementation, SpinDrop.

The evaluations show that the predictive performance of the proposed BayNN with SpinDrop considering variations is also comparable to algorithmic Dropout, as shown in Table 4.8 for MNIST and CIFAR-10 datasets on LeNet-5 and VGG topologies, respectively. We have performed two experiments to evaluate the robustness of the proposed approach to variations in the SpinDrop module. In one case, we trained an NN with normal Dropout, but during Bayesian inference, we evaluated the model against our proposed SpinDrop Dropout with up to 3× the standard deviation σ of the manufacturing variations. In this case, the predictive performance of both MNIST and CIFAR-10 datasets improves slightly (+0.66%) compared to the original algorithmic Dropout BayBNN. Here, the σ of the SpinDrop module is as high as 3.3% (3×). That means the Dropout probability of each neuron can fluctuate by ±10% from the trained value, without any impact on performance. In our experiments, SpinDrop with a dropout probability of 20% is used for all layers and all crossbars.

In the other case, when the NN is trained with the SpinDrop techniques instead of the original Dropout, it would expect Dropout probability $\hat{p} = p + \epsilon, \epsilon \sim \mathcal{N}(\mu, \sigma^2)$. In this case, the prediction performance also is slightly better in comparison, e.g., the accuracy improvement of 1% achieved. Variation in the dropout probability leads to more sparsity during Bayesian inference, and as a result, accuracy improves slightly.

Analysis of Dropout Rate and Location We have used a smaller Dropout rate of 20% in our analysis for both MNIST and CIFAR-10 datasets. The results for other smaller dropout rates, e.g., from 10% to 20% are

Table 4.9.: Predictive performance of proposed SpinDrop BayBNN with different Dropout rates.

Dropout probability	10%	15%	20%
MNIST			
BNN	98.54	98.91	99.09
SpinDrop BayBNN	99.17	99.12	99.27
CIFAR-10			
BNN	87.18	89.08	88.33
SpinDrop BayBNN	88.77	89.74	89.22

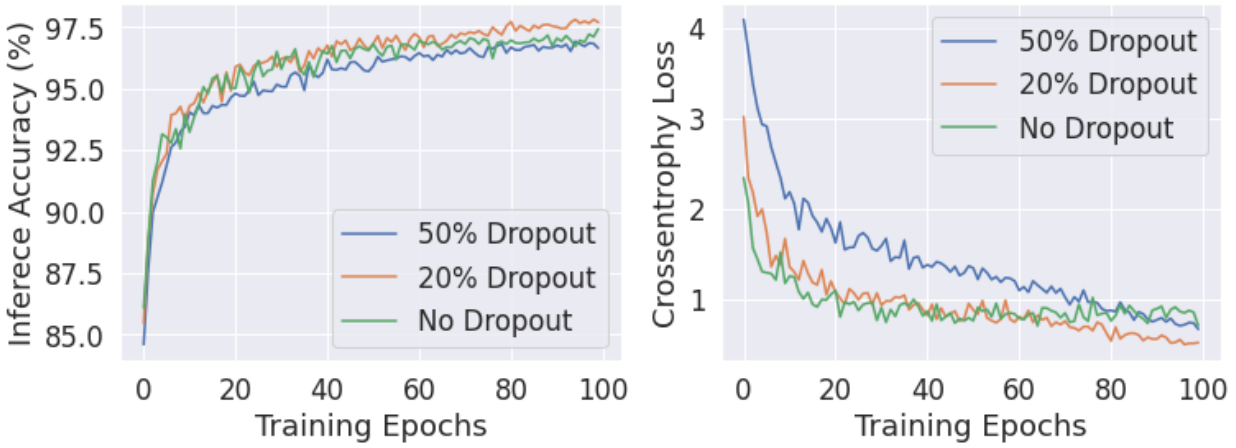


Figure 4.13.: Validation inference accuracy and Cross-entropy loss for 20% and 50% Dropout probability for MNIST on MLP. **It is recommended to view this figure in color.**

summarized in Table 4.9. Here, for each Dropout probability p, we have taken up to 50 MC samples for the Bayesian inference. Our results show that the predictive performance improved for all the Dropout rates by up to 1.7% compared to BNNs.

Usually, the default Dropout probability of 50% is used during training. Although this improves the predictive performance of full-precision NNs, it does not improve the performance of binary NNs, as shown in Figure 4.13. Using 50% Dropout probability achieves lower validation accuracy compared to 20% Dropout probability and no Dropout model. Therefore, we used lower dropout rates in our evaluation for predictive performance.

In all NN topologies and classes, the location of the Dropout layer is very important. In all cases in our analysis, The Dropout layer is applied after Batch Normalization and Sign(.) layers. Otherwise, the dropout effect is canceled. In MLP topology, Dropout was applied in all hidden layers. On the contrary, the location of Dropout in CNN depends on a specific topology. In our experiment with ResNet-18 topology, Dropout is applied to only the last few layers with a large number of parameters. In ResNet-18, applying Dropout to all hidden layers was found to decrease performance. Similar to MLP, Dropout was applied to all hidden layers in the VGG and LeNet-5 topologies.

4.1.3. Scientific Impact of This Work

The scientific impact and contributions of this work can be summarized as follows:

1. **Binary Bayesian Neural Network:** This work introduces Binary Bayesian Neural Networks with an end-to-end approach. Our work demonstrates that binarizing BayNNs can be effectively implemented in a CiM architecture, achieving the benefits of both Bayesian, Binary Neural Networks, and in-memory computation in a single package. Furthermore, the limited stable states challenge of Spintronics has also been overcome. Therefore, research in this direction is highly attractive, making BayNNs more suitable for embedded devices and high-performance applications.

2. **Using Non-Ideality as a Feature:** Our work shows that for unconventional NN, such as stochastic BayNN, non-idealities of memristors can be used as a feature rather than a drawback. Specifically, the inherent stochastic properties of MTJs are leveraged as a feature rather than a drawback. This concept can also be utilized in other memristor technologies for BayNN implementation.
3. **Full Stack Optimization:** Our proposed methods optimize from the algorithmic level to the device level, providing a holistic evaluation and optimization of the BinBayNN. Research in the direction of this approach ensures that all aspects, from training algorithms to circuit-level hardware implementation, are considered and optimized for performance and reliability.
4. **Robustness Of Bayesian Neural Networks:** Our work demonstrates that the proposed BinBayNN can detect out-of-distribution data and is robust against variability and thermal fluctuations that affect MAC results. Our approach shows robustness until 60% of the MC samples. Therefore, research in the direction of BinBayNNs is an attractive solution for reliable predictions. Consequently, such robustness enhances the reliability and safety of BayNN and makes them suitable for safety-critical applications.
5. **Overconfident Predictions of NN:** Our work reinforces the danger of overconfident prediction and shows that even when the data is OOD, a conventional NN makes prediction with high confidence. However, our work addresses the issue of overconfident predictions by incorporating uncertainty estimation into predictions. Therefore, it allows systems to make more informed decisions and avoid blind predictions, thereby increasing the overall reliability and trustworthiness of predictions.
6. **Spintronics Implementation For Low-power BayNN:** We show that by using STT-MRAM, a type of Spintronic memory, a BayNN implementation can achieve high performance and low power consumption. Therefore, the proposed BinBayNN is particularly suitable for resource-constrained and battery-powered devices where power efficiency is critical.
7. **Spintronics Dropout Model:** The proposed model for the Spintronics-based Dropout represents the dropout rate as a distribution to account for process and infield thermal variations. We show that such stochasticity can improve performance. This hardware-based Dropout model contributes to the broader field of deep learning and BayNN implementation.

4.1.4. Section Conclusion

In this section, we introduce the algorithmic groundwork for the Dropout-based Bayesian binary neural network for the first time and the corresponding CiM-based implementation in STT-MRAM. For this purpose, the stochastic and deterministic aspects of STT-MRAM have been combined in a crossbar array-based architecture. The stochastic behavior of the STT-MRAM is leveraged for the implementation of Dropout in hardware required for Bayesian NNs, while the deterministic behavior of STT-MRAM is exploited for the NN weight storage. The results show up to 100% detection capabilities for out-of-distribution data, and up to 15% improvement in accuracy for poisoned data. Furthermore, our results show the high resilience of the proposed concept to process and thermal variations. The combination of the algorithmic Bayesian approach with the cost-effective and energy-efficient implementation of CiM-based Binary NNs allows reliable and also more accurate architectures implementations for deep learning and their usage in critical applications.

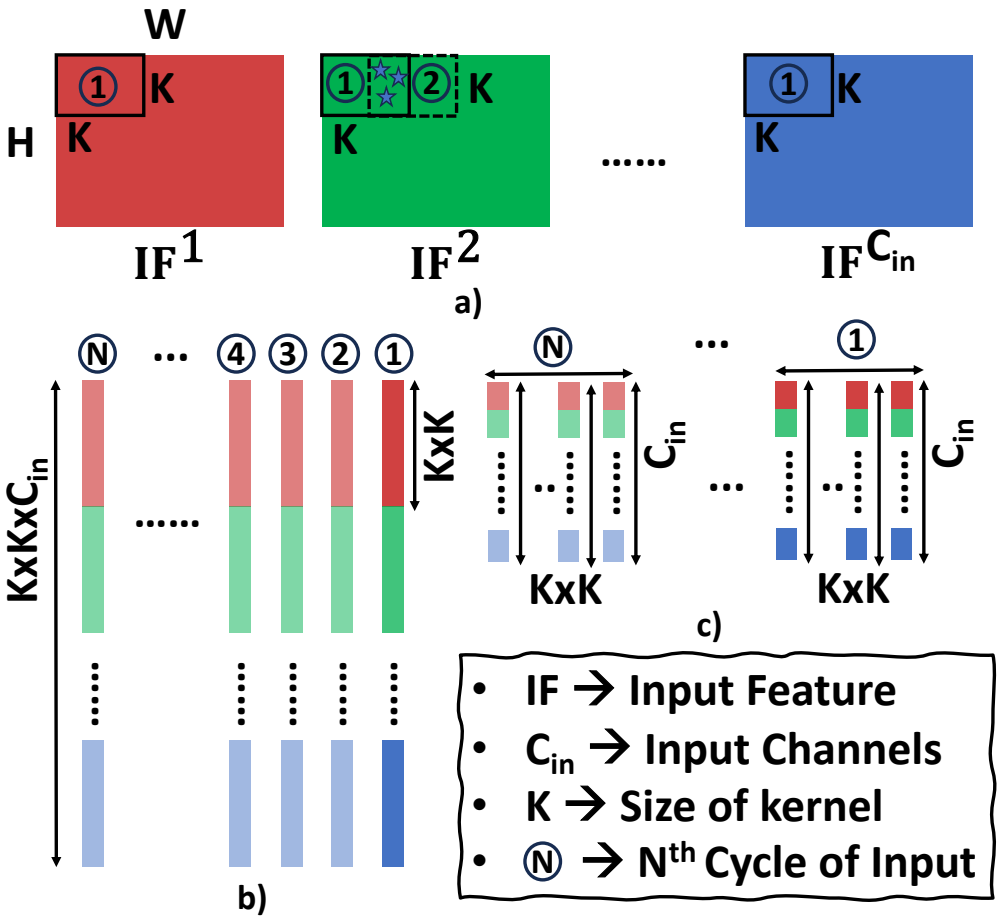


Figure 4.14.: a) Input feature map of a convolutional layer. Moving windows from all the input feature maps are b) flattened and c) parallelized across $k \times K$ crossbars for the conventional mapping. The number circle shows the cycle in which input is applied to the Spintronics crossbar. **It is recommended to view this figure in color.**

4.2. Grouped-Dropout as Bayesian Binary Neural Network

In the last section, although the method based on our papers [84, 85] is highly efficient, it has drawbacks when it comes to convolutional neural networks and the hardware overhead (area and power consumption) related to the Dropout modules. Specifically, integrating the SpinDrop module (proposed in previous work) into CNN requires spatial information to drop certain elements, which complicates circuit design.

In this work, our aim is to address the above-mentioned challenges in an efficient way using the algorithm-hardware co-design approach. This section is based on our journal publication in IEEE TNANO 2024 [86].

4.2.1. Methodology

4.2.1.1. Problem Statement

The convolution operation is performed differently in CiM architectures compared to GPUs. In CiM architectures, moving windows (MWs) with a shape of $K \times K$ are applied to each input feature map (IFM) in one cycle (see Fig. 4.14(a)). In the next cycle, the MWs will "slide over" the IFMs with a topology-defined stride S for N cycles. Assuming $K > S$, some of the elements in the MWs for the next $K - S$ cycles will be the same as in the previous cycles, a concept known as weight sharing. This is illustrated by the green input feature (IF) in Fig. 4.14(a). The NN topology and the task determine the size of the stride and the

moving windows, which are hyperparameters that the user cannot modify online, i.e., during mapping. On the other hand, the problem can be mitigated by training the CNN with stride $S \geq K$, but it can reduce performance as a trade-off.

The SpinDrop module designed in the previous work [84, 85] drops each element of the MWs with a probability P in each cycle. Therefore, integrating it into convolutional layers implemented in CiM results in essentially re-sampling the dropout mask of each MW of IFMs in each cycle. Consequently, the dropout masks of the shared elements in the MWs will change in each input cycle, leading to inconsistency. An ideal Dropout module should only generate dropout masks for new elements of the MWs. Designing a Dropout module that drops each element of the MWs depending on the spatial location of the MWs in the IFMs is challenging and may lead to complex circuit design. Additionally, the number of rows in crossbars typically increases from one layer to another due to the larger C_{in}. Consequently, the number of Dropout modules required will be significantly higher as the SpinDrop module is applied to each row of the crossbar.

Furthermore, the MWs are reshaped depending on the weight mapping discussed in Section 2.9.1. For the mapping strategy ①, the MWs of the IFMs are flattened into a vector of length $K \times K \times C_{in}$. However, for the mapping strategy ②, IFMs are flattened into $K \times K$ vectors of length C_{in}, as depicted in Fig. 4.14(a) and (b). As a result, designing a generalizable Dropout model is challenging.

Furthermore, in terms of computational costs and performance requirements of BayNNs on edge devices, real-world applications require high throughput, low area, and energy consumption. The dropout module introduces additional power consumption and area, which increase linearly with the number of dropout modules. Consequently, this makes it challenging to deploy BayNNs on edge devices with limited resources and introduces the resource scalability issue.

4.2.1.2. MC-SpatialDropout as Bayesian Approximation

In an effort to improve the efficiency and accuracy of Bayesian approximation techniques in CiM, we propose to drop groups of word-lines in the crossbar together. In the literature several group-wise dropouts are proposed, such as Spatial-Dropout [56]. However, instead of proposing a new Dropout method, we chose to use spatial Dropout and propose *binary* MC-SpatialDropout method. However, it should be noted that our approach is not tied to Spatial-Dropout. Since our objective is to drop a group of word-lines together, any group-wise application of Dropout would work in this context.

Therefore, the proposed *binary* MC-SpatialDropout technique extends on the MC-Dropout [35], MC-SpatialDropout [113], and MC-SpinDrop [84, 85] methods by utilizing spatial dropout as a Bayesian approximation in binary NNs and Spintronics implementation. Our approach drops an entire feature with a probability p. This means that all the elements of a feature map in Fig. 4.14(a) are dropped together. However, each feature map is dropped independently of the others. As a result, the number of Dropout modules required for a layer will be significantly reduced, and the design effort of the dropout module will also be lessened.

The primary objective of this approach is to address the shortcomings of MC-SpinDrop [84, 85] that arise from its independent treatment of elements of the features. In contrast, MC-SpatialDropout exploits the spatial correlation of IFs, which is particularly advantageous for tasks that involve image or spatial data. In doing so, it facilitates a more robust and contextually accurate approximation of the posterior distribution. This enables the model to capture more sophisticated representations and account for dependencies between features.

In terms of the objective function for the MC-SpatialDropout, our previous work [84, 85] showed that minimizing the objective function of MC-Dropout (see Equation equation 2.10) is not beneficial for BNNs

and suggested a BNN-specific regularization term. In this paper, instead of defining a separate loss function for MC-SpatialDropout, we define the objective function as:

$$\mathcal{L}(\theta)_{\text{MC-SpatialDropout}} = \mathcal{L}(\theta, \mathcal{D}) + \lambda \sum_{l=1}^{L} ||\mathbf{W}_l||_2^2. \tag{4.9}$$

Therefore, the objective function is equivalent to Equation equation 2.10 for MC-Dropout. However, the second part of the objective function is the regularization term applied to the (real-valued) "proxy" weights ($\mathbf{W}$) of BNN instead of binary weights. It encourages $\mathbf{W}$ to be close to zero. By keeping a small value for the λ, it implicitly ensures that the distribution of weights is centered around zero. Also, we normalize the weights by

$$\bar{\mathbf{W}}_l = \frac{\mathbf{W}_l - \mu_l^{\mathbf{W}}}{\sigma_l^{\mathbf{W}}}, \tag{4.10}$$

to ensure, the weight matrix has zero mean and unit variance before binarization. Where $\mu^{\mathbf{W}}$ and $\sigma^{\mathbf{W}}$ are the mean and variance of the weight matrix of the layer l. This process allows applying L2 regularization in BNN training and existing work [47] showed that it improves inference accuracy by reducing quantization error. Since our work is targeted for BNN, regularization is only applied to the weight matrixes.

The difference is that our method approximate Equation 2.8 by:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}|\mathbf{x}, \theta, \hat{\mathbf{M}}_t) \quad \text{with} \quad \hat{\mathbf{M}}_t \sim \mathcal{B}(p). \tag{4.11}$$

Here, during training and Bayesian inference, the dropout mask $\hat{\mathbf{M}}_t$ sampled spatially correlated manner for the output feature maps (OFMs) of each layer from a Bernoulli distribution with (dropout) probability p. The dropout masks correspond to whether a certain spatial location in the OFMs (i.e., a certain unit) is dropped or not.

Proper arrangement of layers is important for the MC-SpatialDropout-based Bayesian inference. The Spatial Dropout layer can be applied before each convolutional layer in a layerwise MC-SpatialDropout method. Additionally, the Spatial Dropout layer can be applied to the extracted features (of the last convolutional layer) of a CNN topology in a topology-wise MC-SpatialDropout method. For Bayesian inference and uncertainty, we perform T Monte Carlo sampling similar to [35].

4.2.1.3. Designing Spatial-SpinDrop Module

As mentioned earlier, in the proposed MC-SpatialDropout implementation, feature maps can independently be dropped with a probability p. Due to the nature of input application in CiM architectures, this implicitly means dropping different regions of crossbars depending on the mapping strategy. These challenges are associated with designing the Dropout module for the proposed MC-SpatialDropout based BayNN. Consequently, we propose four different Dropout module designs.

For the mapping strategy ①, as depicted in Fig. 4.14(b), each $K \times K$ subset of input comes from a feature map. This means that if an input feature is dropped, the corresponding $K \times K$ subset of the input should also be dropped for all C_{out} and N cycles of inputs. This implies that dropping each $K \times K$ row of a crossbar together for N cycles is equivalent to applying spatial dropout. However, each group of rows should be dropped independently of one another. Additionally, their dropout mask should be sampled only in the first cycle. For the remaining $N - 1$ cycles of input, the dropout mask should remain consistent.

In contrast, in the mapping strategy ② (see Fig. 4.14(c)), the elements of a MW are applied in parallel to each $K \times K$ crossbar at the same index. As a result, dropping an IF would lead to dropping each index of rows in all the $K \times K$ crossbars together. Similarly, each row of a crossbar is dropped independently of one

another, and the dropout mask is sampled at the first input cycle and remains consistent for the remaining $N-1$ cycles of input.

Furthermore, if the spatial dropout is applied to the extracted feature maps of a CNN, then depending on the usage of the adaptive average pool layer, the design of the Spin-SpatialDrop will differ. If a CNN topology does not use an adaptive average pool layer but the spatial Dropout is applied to the last convolutional layer, then $H \times W$ groups of rows are dropped together. This is because the flattening operation essentially flattens each IF into a vector. These vectors are combined into a larger vector representing the input for the classifier layer. However, since input for the FC layer is applied in one cycle only, there is no need to hold the dropout mask. The Spin-SpatialDrop module for the mapping strategy ① can be adjusted for this condition.

Lastly, if a CNN topology does use an adaptive average pool layer, then the SpinDrop module proposed by [84, 85] can be used. This is because the adaptive average pool layer averages each IF to a single point, giving a vector with a total of C_{out} elements.

Therefore, the Dropout module for the proposed MC-SpatialDropout should be able to work in four different configurations. Consequently, we propose a novel spintronic-based spatial Dropout design, called *Spatial-SpinDrop*.

The Spatial-SpinDrop module leverages the stochastic behavior of the MTJ for spatial dropout. The proposed scheme is depicted in Fig. 4.15. In order to generate a stochastic bitstream using the MTJ, the first step involves a writing scheme that enables the generation of a bidirectional current through the device. This writing circuit consists of four transistors, allocated to a "SET" and a "RESET" modules. The "SET" operation facilitates the stochastic writing of the MTJ, with a probability corresponding to the required dropout probability. On the other hand, the "RESET" operation restores the MTJ to its original state. During the reading operation of the MTJ, the resistance of the device is compared to a reference element to determine its state. The reference resistance value is chosen such as it falls between the parallel and anti-parallel resistances of the MTJ.

For the reading phase, a two-stage architecture is employed for better flexibility and better control of the reading phase for the different configurations discussed earlier. The module operates as follows: after a writing step in the MTJ, the signal V_{pol} allows a small current to flow through the MTJ and the reference cell (*REF*), if and only if the signal *hold* is activated. Thus, the difference in resistance is translated into a difference in voltages (V_{MTJ} and V_{ref}). The second stage of the amplifier utilizes a StrongARM latch structure [174] to provide a digital representation of the MTJ state. The *Ctrl* signal works in two phases. When *Ctrl = 0*, $\overline{Out}$ and *Out* are precharged at *VDD*. Later, when *Ctrl = 1*, the discharge begins, resulting in a differential current proportional to the gate voltages (V_{MTJ} and V_{ref}). The latch converts the difference of voltage into two opposite logic states in $\overline{Out}$ and Out. Once the information from the MTJ is captured and available at the output, the signal *hold* is deactivated to anticipate the next writing operation. To enable the dropout, a series of AND gates and transmission gates are added, allowing access to either the classical decoder or to the stochastic word-line (WL).

As long as the *hold* signal is deactivated, no further reading operation is permitted. Such a mechanism allows the structure to maintain the same dropout configuration for a given time and will be used during $N-1$ cycles of inputs to allow the dropping of the IF in strategies ① and ②. In the first strategy, the AND gate receives as input $K \times K$ WLs from the same decoder, see Fig. 4.16(a). While in strategy ②, the AND gate receives one row per decoder, as presented in Fig. 4.16(b).

For the last two configurations, the *hold* signal is activated for each reading operation, eliminating the need to maintain the dropout mask for $N-1$ cycles.

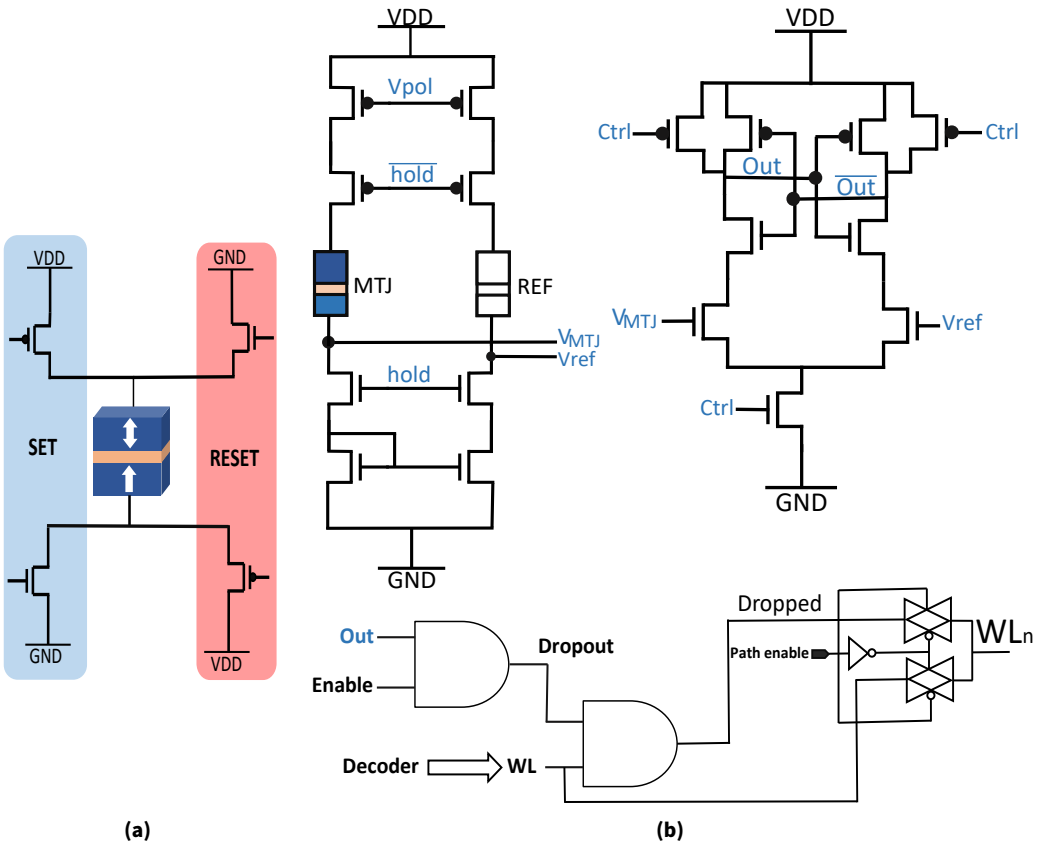


Figure 4.15.: (a) Writing and (b) reading schemes for the MTJ.

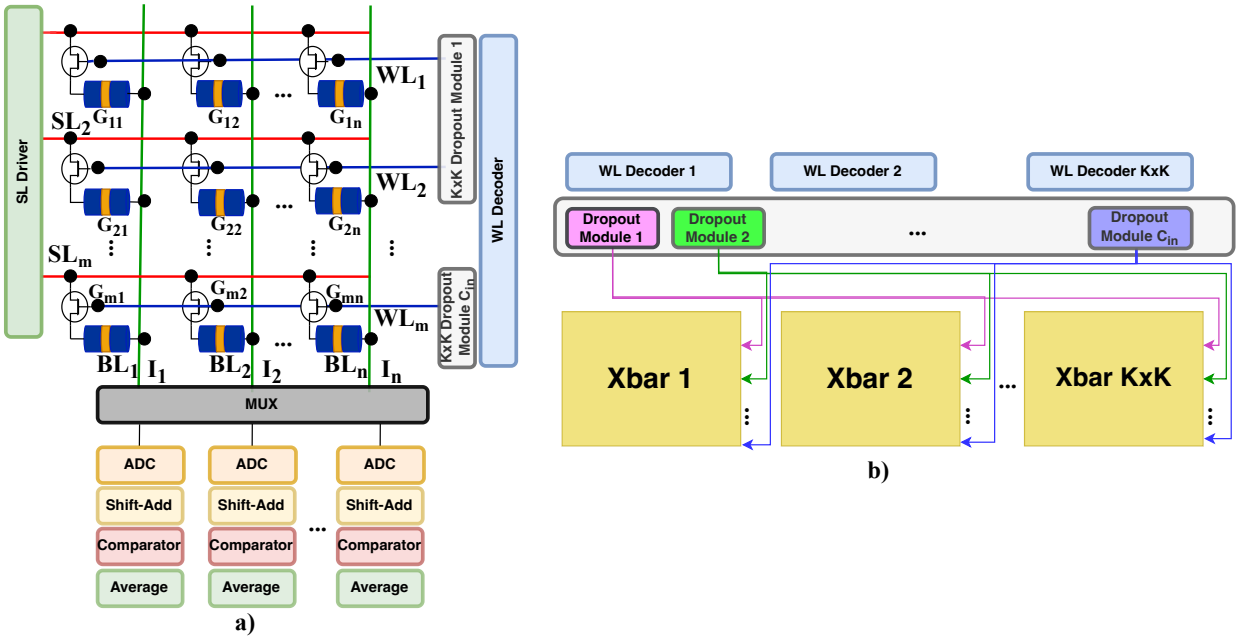


Figure 4.16.: Crossbar design for the MC-SpatialDropout based on mapping strategy (a) ① and (b) strategy②. In (b), only the Dropout module and WL decoder are shown. Everything else is abstracted.

4.2.1.4. MC-SpatialDropout-Based Bayesian Inference in CiM

The proposed MC-Spatial Dropout-Based Bayesian inference can be leveraged on the two mapping strategies discussed in Section 2.9.1. In both strategies, one or more crossbar arrays with MTJs at each cross point are employed in order to encode the binary weights into the resistive states of the MTJs.

Specifically, for the mapping strategy ①, we divide the WLs of the crossbar into $K \times K$ groups and connect one dropout module to each group, as shown in Fig.4.16(a). In Fig. 4.15(b), this strategy involves connecting $K \times K$ WLs to an AND gate. The AND gate receives the signal delivered by the decoder as its input. This configuration allows for the selective activation or deactivation of a group of WLs. To facilitate the activation of multiple consecutive addresses in the array, an adapted WL decoder is utilized. The bit-line and source-line drivers were used to manage the analog input and output for MVM operation. Also, a group-wise selection of WLs is performed concurrently, and the intermediate result for MVM operation is accumulated into an accumulator block until all the WLs are selected for each layer. We utilized MUXes to select the different bit-lines that are sensed and converted by ADC. The shift-adder modules are used to shift and accumulate the partial sums coming from the array. Finally, a digital comparator and averaging block are used to implement the activation function. For the last layer, the average operation is performed with an averaging block.

For the mapping strategy ②, a similar architecture to the strategy ① is employed. The key distinction relies upon the utilization of $K \times K$ crossbars in parallel to map the binary weights of a layer. Also, the dropout modules are connected to a similar WL index in each of the crossbar arrays, as shown in Fig. 4.16(b). Here, the same AND gate in the Dropout module receives signals from different decoders and the result is sent to each row of the $K \times K$ crossbars. For instance, the first WL of each crossbar of a layer connects the same Dropout module. All the WLs decoders are connected to a dropout block in gray in Fig. 4.16(b) comprising C_{in} dropout modules. It is worth mentioning that the dropout is used during the reading phase only, therefore, the dropout module is deactivated during the writing operation and WL decoders are used normally.

4.2.2. Results

4.2.2.1. Evaluation Setup

We evaluated the proposed MC-SpatialDropout on predictive performance in a) image classification task using VGG, ResNet-18, and ResNet-20 topologies in the CIFAR-10 dataset, b) biomedical semantic segmentation task using a more complex U-Net topology in digital retinal images for vessel extraction (DRIVE) dataset. All models were trained with the SGD optimization algorithm, minimizing the proposed learning objective equation 4.9 with λ chosen between 1×10^{-5} and 1×10^{-7}, and the binarization algorithm from [47] was used. Furthermore, all models are trained with $p = 15\%$ dropout probability. We have used SGD due to its proven effectiveness in training, particularly for large-scale data, and its ability to efficiently converge to optimal solutions. Other optimization algorithms, such as ADAM and the binarization algorithm that uses weight normalization, as discussed in the methodology section of this work, would also work for our approach.

To assess the effectiveness of our method in handling uncertainty, we generated six additional OOD datasets: 1. Gaussian noise ($\hat{\mathcal{D}}_1$), 2. Uniform noise ($\hat{\mathcal{D}}_2$), 3. CIFAR-10 with Gaussian noise ($\hat{\mathcal{D}}_3$), 4. CIFAR-10 with uniform noise ($\hat{\mathcal{D}}_4$), 5. SVHN: Google street view house numbers dataset, and 6. STL10: a dataset containing images from the popular ImageNet dataset. Each of these OOD datasets contains 8000 images, and the images have the same dimensions as the original CIFAR-10 dataset. During the evaluation phase,

$$\begin{cases} \text{OOD}, & \text{if } \max\left(\mathcal{G}\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{y}_t\right)\right) < 0.9 \\ \text{ID}, & \text{otherwise.} \end{cases} \tag{4.12}$$

Here, $\mathbf{y}_t$ is the softmax output of the stochastic forward pass on the MC run t with T overall MC runs, the function $\mathcal{G}(\cdot)$ calculates the 10th percentile across a set of values, and the function $\max(\cdot)$ determines the maximum confidence score across output classes. In general, OOD or ID is determined by whether the maximum value from the 10th percentile of the averaged outputs is less than 0.9 (for OOD) or not (for ID). The intuition behind our OOD detection is that the majority of confidence scores of the T MC runs are

Table 4.10.: Predictive Performance of the proposed MC-SparialDropout method in comparison with SOTA methods on CIFAR-10.

Topology	Method	Bit-width (W/A)	Bayesian	Inference Accuracy
ResNet-18	FP	32/32	No	93.0%
	RAD [175]	1/1	No	90.5%
	IR-Net [47]	1/1	No	91.5%
	SpinDrop [84, 85]	1/1	Yes	90.48%
	Proposed	1/1	Yes	**91.34%**
ResNet-20	FP	32/32	No	91.7%
	DoReFa [176]	1/1	No	79.3%
	DSQ [177]	1/1	No	84.1%
	IR-Net [47]	1/1	No	85.4%
	Proposed	1/1	Yes	**84.71%**
VGG	FP	32/32	No	91.7%
	LAB [178]	1/1	No	87.7%
	XNOR [45]	1/1	No	89.8%
	BNN [44]	1/1	No	89.9%
	RAD [175]	1/1	No	90.0%
	IR-Net [47]	1/1	No	90.4%
	SpinDrop [84, 85]	1/1	Yes	91.95%
	Proposed	1/1	Yes	**90.34%**

expected to be high and close to each other (low variance) for ID data and vice versa for OOD data, as discussed in the previous work and depicted by [35]. On the other hand, for hardware-level simulation, the 28nm FDSOI ST-Microelectronics technology was used to perform simulations on the Cadence Virtuoso simulator.

4.2.2.2. Predictive Performance and Uncertainty Estimation

The predictive performance of the approach is comparable to the existing conventional BNNs, as shown in Table 4.10. Furthermore, compared to Bayesian approaches [84, 85], our proposed approach is within 1% inference accuracy. Furthermore, the application of Spatial-SpinDrop before the convolutional layer and in the extracted feature maps can also achieve comparable performance ($\sim$ 0.2%). Also, for biomedical semantic segmentation, the prediction mask for the proposed method roughly resembles the ground truth label and is similar to full precision MC-Dropout [35]. In terms of intersection over union (IOU) score and pixel accuracy, our approach can achieve a 64.60% IOU score and a pixel accuracy of 95.97%. In comparison, the SpinDrop [85] method has 96.49% pixel accuracy and 67.44% IOU, which is comparable. This demonstrates the capability of the proposed approach to achieve high predictive performance. However, note that applying Spatial-SpinDrop before all the convolutional layers can drastically reduce the performance; e.g., the accuracy reduces to 75% on VGG. This is because at shallower layers, the number of OFMs is lower compared to that at deeper layers, leading to a high chance that most OFMs are being omitted (dropped). Furthermore, as shown by [84, 85], BNNs are more sensitive to the dropout rate. Therefore, a lower Dropout probability between $10 - 20\%$ is suggested.

In terms of OOD detection, our proposed method can achieve up to 100% OOD detection rate across various model architectures and six different OOD datasets ($\hat{\mathcal{D}}_1$ through $\hat{\mathcal{D}}_6$), as depicted in Table 4.11. There are some variations across different architectures and OOD datasets. However, even in these cases, our method can consistently achieve a high OOD detection rate, with the lowest detection rate being 64.39% on the

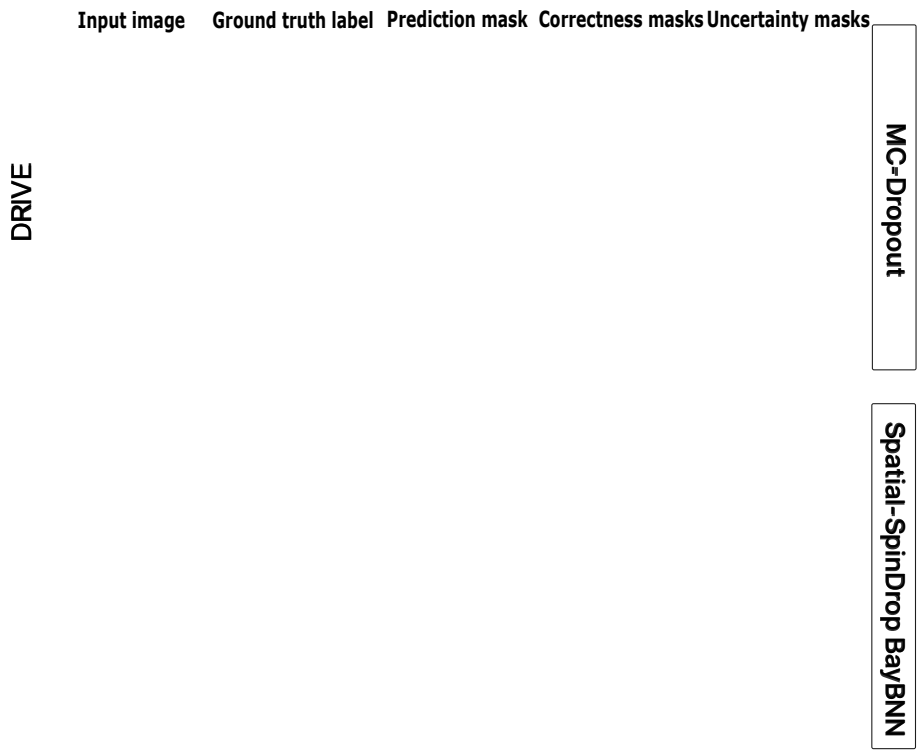


Figure 4.17.: Results of semantic segmentation and estimations of uncertainty for the DRIVE. The correctness map is a binary representation of correct and incorrect predictions. The uncertainty map is the normalized [0, 1] map of uncertainty values derived after 20 Monte Carlo samples.

Table 4.11.: Evaluation of the proposed MC-SpatialDropout method in detecting OOD.

Topologie	$\hat{\mathcal{D}}_1$	$\hat{\mathcal{D}}_2$	$\hat{\mathcal{D}}_3$	$\hat{\mathcal{D}}_4$	$\hat{\mathcal{D}}_5$	$\hat{\mathcal{D}}_6$
ResNet-18	99.56%	99.94%	96.1%	81.68%	83.02%	64.39%
ResNet-18[i]	100%	100%	100%	92.26%	99.98%	97.39%
ResNet-20	97.2%	100%	90.79%	87.94%	99.03%	99.81%
VGG	99.99%	100%	92.9%	78.91%	99.81%	100%

[i] Spatial-Dropout applied to the final two convolutional layers.

ResNet-18 model with the $\hat{\mathcal{D}}_4$ dataset and Spatial-SpinDrop applied to extracted feature maps. However, when Spatial-SpinDrop is applied to the convolutional layers of the last residual block, the OOD detection rate on the $\hat{\mathcal{D}}_4$ dataset improved to 97.39%, a 33.00% improvement. This is because Bayesian treatments are applied to more parameters (i.e., weights are probabilistic rather than fixed) and the model regularizes more layers. Thus, stronger uncertainty estimates can be obtained. Therefore, we suggest applying Spatial-SpinDrop to the last convolutional layers to achieve a higher OOD detection rate at the cost of a small accuracy reduction. In terms of biomedical semantic segmentation, our approach provides an ideal or close-to-ideal uncertainty mask. Specifically, the uncertainty is high around incorrectly predicted pixels and low around correctly predicted pixels. Consequently, the result suggests that the MC-SpatialDropout method is a robust and reliable approach to OOD detection in various model architectures and datasets.

4.2.2.3. Overhead Analysis

The proposed Spatial-SpinDrop modules were evaluated for area, power consumption, and latency as shown in Table 4.12 and compared with the SpinDrop approach presented in [84, 85]. These evaluations were performed using a crossbar array with dimensions of 64×32 and scaled for the larger VGG topology from LeNet-5. In the layer-wise application of spatial Dropout, the Dropout modules were applied to the convolutional layers of the last VGG block. Also, for topology-wise application of spatial Dropout, Dropout

Table 4.12.: Layer-wise Overhead Analysis of the Proposed Method in Comparison to SpinDrop [84, 85].

Layer-wise application of spatial Dropout					
Method	Mapping Strategy	# of Dropout Modules	Area	Power Consumption	Sampling Latency
SpinDrop	①	$K*K*C_{in}$	$79833.6\mu m^2$	$51.84mW$	$15ns$
	②	$K*K*C_{in}$	$79833.6\mu m^2$	$51.84mW$	
Proposed	①	C_{in}	$8870.4\mu m^2$	$5.76mW$	
	②	C_{in}	$8870.4\mu m^2$	$5.76mW$	
Topology-wise application of spatial Dropout					
Method	Adaptive Avg. Pool	# of Dropout Modules	Area	Power Consumption	Sampling Latency
SpinDrop	Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	$15ns$
	Not Used	$K*K*C_{out}$	$159667.2\mu m^2$	$103.68mW$	
Proposed	Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	
	Not Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	

Table 4.13.: Energy Efficiency Comparison of Hardware Implementations

Related works	Technology	Dataset	Topology	Bit resolution	Energy
H. Fan et al.[96]	FPGA	CIFAR-10	ResNet18	8-bit	0.014 J/Image
R.Cai et al.[168]	FPGA	MNIST	3-FC	8-bit	18.97 µJ/Image
X.Jia et al.[169]	FPGA	MNIST	3-FC	8-bit	46.00 µJ/Image
H.Awano et al. [95]	FPGA	MNIST	3-FC	7-bit	21.09 µJ/Image
A. Malhotra et al. [99]	RRAM	MNIST	3-FC	4-bit	9.30 µJ/Image
S.T.Ahmed et al.[84]	STT-MRAM	MNIST	LeNet-5	1-bit	2.00 µJ/Image
K.Yang et al.[102]	Domain wall-MTJ	MNIST	3-FC	4-bit	0.79 µJ/Image
Proposed implementation	**STT-MRAM**	MNIST	**3-FC**	**1-bit**	**0.12** µJ/Image
Proposed implementation	**STT-MRAM**	MNIST	**LeNet-5**	**1-bit**	**0.68** µJ/Image
Proposed implementation	**STT-MRAM**	CIFAR-10	**VGG**	**1-bit**	**1.31** µJ/Image

modules are applied to the extracted feature maps. In our evaluation, a configuration of $C_{in} = 256$, $K = 3$ and $C_{out} = 512$ is used.

At first, in terms of area, the SpinDrop method requires one dropout module per row in the crossbar structure, while our method only requires one dropout module per $K \times K$ group of rows. Therefore, the area and the power consumption of the dropout modules are reduced by a factor of K^2, which for VGG is 9. In terms of latency for the dropout modules, we achieve $15ns$ in all cases. Indeed, to generate 1 bit, for a given number of rows, the dropout module needs to be written. However, such latency can be further reduced by increasing the writing voltages of the MTJ, but at the cost of higher power consumption. Furthermore, when the adaptive average pool layer is not used, the power consumption and the area of the SpinDrop approach increase greatly (×9). However, in the proposed approach, the adaptive average pool layer does not impact total energy and area, as mentioned in Section 4.2.1.3 and shown in Table 4.12.

Table 4.13 compares the energy consumption of the proposed approach with the state-of-the-art implementation based on the MNIST and CIFAR-10 datasets. For the evaluation, we used NVSIM and estimated the total energy of a 3-FC, LeNet-5, and VGG architectures. To scale our approach, we also estimate the energy consumption for the CIFAR-10 dataset. Compared to existing spintronic implementations, we achieve savings of 6.58× compared to [102], and 2.94× savings compared to the SpinDrop approach in [84], with respect to the architecture. Furthermore, compared to RRAM technology, our solution is 77.5× more efficient. Finally, compared to the classic FPGA implementation, the proposed approach achieves up to 300× more energy savings.

4.2.2.4. Discussion

This work focuses on the implementation of binary Bayesian neural networks with Spintronic technology. However, stochastic and deterministic features also exist in other technologies, such as PCM [179] or RRAM [99]. Compared to RRAM and PCM, for which stochasticity is due to filament creation and the random nature of the phase change process, the stochastic behavior of STT is obtained for low voltage, and thus allows us to save more energy [84, 85].

Hardware-level simulations, conducted with the Cadence simulator, evaluate the performance of the proposed MC-SpatialDropout method. The results show an improvement in both area efficiency and energy consumption, emphasizing the resilience of the approach. However, for practical deployment, it is crucial to carefully consider design factors and challenges. A critical consideration is the impact of the IR drop on the size of the array. Indeed, adjusting the array size to match the specific requirements of the application is essential, and the design may be influenced by the phenomenon of IR drop. Variability presents another significant concern that requires a thorough evaluation, as it affects not only hardware design, but also system-level development and algorithmic implementation.

The impact of process variations and mismatches for different dropout rates has been evaluated, and the approach demonstrates robustness and resilience in the face of variability [84, 85]. Furthermore, the variability of the crossbar was examined to illustrate the influence of stochasticity and device variability. These assessments validate the effectiveness of our approach and enable scalability to larger architectures.

Moving to the system level, appropriately resizing the array to minimize the effects of variability and IR drop for efficient computation enables complete digitalization of the system. This transforms it into a mapping problem for a larger neural network architecture. The mapping of neural network layers is discussed in Sections 4.2.1.1 and 4.2.1.3. Indeed, in our work, effective implementation techniques were shown to integrate the dropout method for both FC and convolutional layers into CiM architectures. Regarding scalability in the context of dropout application, there is a discernible linear relationship between metrics such as area and power consumption and the depth of layers. This relationship is specifically related to the number of output channels (C_{out}) of the last convolutional layers in the topology-wise application and the input channels (C_{in}) of the convolutional layers in the layer-wise application of spatial Dropout.

To achieve improved accuracy regardless of our hardware setup, it is necessary to use more advanced neural network architectures with a larger number of neurons and layers. For CIFAR-10, the VGG topology was preferred. However, selecting a simpler architecture, such as LeNet-5, would improve energy efficiency but reduce accuracy as a trade-off. Conversely, a more complex architecture, e.g., ResNet, would greatly enhance accuracy but reduce energy efficiency. In edge applications with limited power budgets, maximizing accuracy isn't the main priority; a balance between accuracy and energy efficiency is crucial. Therefore, it is essential to carefully select the most suitable model to achieve the highest accuracy within a limited power budget.

In this work, we introduce a novel method for implementing convolutional layers within the Bayesian framework, irrespective of the chosen architecture. Our approach efficiently incorporates dropout on convolutional layers, enhancing power efficiency across various architectures. However, it is important to note that the choice of architecture for a given application must be made, given the trade-offs discussed earlier.

4.2.3. Scientific Impact of This Work

We outline the scientific impact of this work and our main contributions:

1. **Group-wise Dropout and Resource Scalability:** This work propose to apply Dropout in a group-wise manned in CiM architecture. Consequently, the energy consumption and chip area can be reduced by $k^2\times$ when Dropout is applied to $k \times k$ groups of wordlines together. Therefore, research in the direction of group-wise can improve resource scalability.

2. **Bayesian Treatment of Parameters:** The more parameters are treated as Bayesian or probabilistic, the better the uncertainty estimates. However, the accuracy can be reduced as a trade-off.
3. **Adaptive Dropout In CiM Architecture:** Four different Dropout configurations and their respective CiM architecture depending on the weight mapping method are proposed.
4. **Real-World Applications:** The work showed improvements in energy efficiency and performance metrics while reducing the chip area and achieving a high detection capability of the OOD data. Thus, making it suitable for real-world applications, especially in domains requiring real-time processing and decision-making under uncertainty.

4.2.4. Section Conclusion

In this section, we present grouped Dropout for Bayesian inference in CiM architecture. We implement the grouped Dropout concept using spatial Dropout, which drops each spatial feature map with a probability. Furthermore, we propose the MC-SpatialDropout is an efficient spatial dropout-based approximation for Bayesian neural networks in CiM architecture. The proposed method exploits the probabilistic nature of spintronics technology to enable Bayesian inference. Implemented on a spintronics-based Computation-in-Memory fabric with STT-MRAM, MC-SpatialDropout achieves improved computational efficiency and power consumption while maintaining performance and quality of uncertainty estimates.

4.3. Scale Dropout-Based Bayesian Binary Neural Network

In the previous section that is based on our publication [86], we realized that applying Dropout in a group-wise manner can improve resource scalability. Resource scalability is important for deploying BayNNs in a resource-constrained device for edge AI applications.

When it comes to improving resource scalability with group-wise dropout, the extreme end of this is when all the word lines of a crossbar are dropped together. Consequently, **a single Dropout module is required per layer** of BayNN in a CiM architecture. It leads to a reduction in the number of Dropout modules and associated chip area and power by a factor of up to $C_{out} \times C_{in} \times K_h \times K_w$ for a single layer compared to our previous works [85, 84, 86]. This further reduces by up to $L\times$ when Dropout is applied to L layers. However, in this case, this results in total loss of information, as it is equivalent to performing MAC operations in a layer l with zero inputs.

Therefore, a different strategy is required, compared to the conventional dropout or group-wise dropout. In this work, we propose a novel dropout method in which a specific parameter group of binary NNs, the scale vector, is dropped for uncertainty estimation. The work is based on our publication [49].

4.3.1. Scale Dropout

4.3.1.1. Scale Vector

To reiterate Section 2.2.1, the scale vector is crucial in BNN to reduce the quantization error, but in CiM, scaling of the weight matrix is not feasible due to the limited stable states of the Spintronic device. Therefore, we propose a hardware-software co-design approach for the scaling factor in BNN that is suitable for CiM architecture. Specifically, we design our scale factor (denoted as $\boldsymbol{\alpha}$) to be learnable through a gradient descent algorithm and the same shape as the bias vector of a layer, $\boldsymbol{\alpha} \in \mathbb{R}^{C_{out} \times 1 \times 1 \times 1}$. Here, C_{out} represents the number of output channels in the convolutional layers and the number of neurons in the linear layers. This choice is motivated by the desire to reduce memory overhead while ensuring compatibility with the CiM architecture. By making the scale factor learnable, we allow the training process to determine

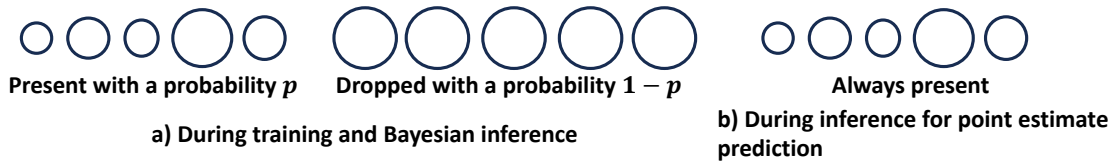


Figure 4.18.: Several nodes (neurons) a) at training time that are scaled with a probability of p and dropped (ignored) with a probability of $1-p$, b). At test time, if point estimate prediction is preferred, all the nodes are always scaled. However, for Bayesian inference, all nodes behave similarly to train time. Here, all the nodes are connected to the weights of the next layer after non-linear activation and Batch normalization, and their shapes represent scaling factors. All the dropped nodes have the same shape, indicating no scaling factor.

the optimal scale factor, making the model more adaptive and possibly improving its performance [46]. Note that the learnable parameters and the two variables (μ and σ) of the batch normalization layer have the same shape as the bias vector of a layer. Therefore, choosing the same shape of scale vector as those vectors leads to simplified computation and storage in the CiM architecture.

4.3.1.2. Scale Dropout Model Description

Let a BNN with L hidden layers and $\mathbf{z}^{(l-1)}$ denote the input vector, $\mathbf{z}^{(l)}$ denotes the output vector, $\boldsymbol{\alpha}^{(l)}$ denotes the scale vector, $\mathbf{W}^{(l)}$ denotes the weights and $\mathbf{b}^{(l)}$ denotes the biases of the layer l. The feed-forward operation (for $l = 1, \cdots, L-1$) of BNN can be described as

$$\mathbf{z}^{(l)} = (\text{sign}(\mathbf{W}^{(l)})^{\top} \otimes \text{sign}(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}) \odot \boldsymbol{\alpha}^{(l)} \tag{4.13}$$

$$\hat{\mathbf{z}}^{(l)} = \text{BatchNorm}_{\gamma,\beta}(\mathbf{z}^{(l)}) \tag{4.14}$$

$$\bar{\mathbf{z}}^{(l)} = \phi(\hat{\mathbf{z}}^{(l)}) \tag{4.15}$$

Where ϕ denotes the element-wise nonlinear activation function for BNN, e.g., the Tanh$(\cdot)$ function (hyperbolic Tangent), $\top$ denotes the matrix transpose operation and BatchNorm$_{\gamma,\beta}(\cdot)$ denotes the bach normalization [50] with a learnable parameter γ and β. In addition, $\odot$ denotes element-wise multiplication, and $\otimes$ denotes binary convolution. With Scale-Dropout, the feed-forward operation becomes :

$$\mathcal{M}^{(l)} \sim \text{Bernoulli}(p) \tag{4.16}$$

$$\hat{\boldsymbol{\alpha}}^{(l)} = \boldsymbol{\alpha}^{(l)} \cdot \mathcal{M}^{(l)} \tag{4.17}$$

$$\mathbf{z}^{(l)} = (\text{sign}(\mathbf{W}^{(l)})^{\top} \otimes \text{sign}(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}) \odot \hat{\boldsymbol{\alpha}}^{(l)} \tag{4.18}$$

$$\hat{\mathbf{z}}^{(l)} = \text{BatchNorm}_{\gamma,\beta}(\mathbf{z}^{(l)}) \tag{4.19}$$

$$\bar{\mathbf{z}}^{(l)} = \phi(\hat{\mathbf{z}}^{(l)}) \tag{4.20}$$

Here, the Dropout mask for the scale Dropout is defined as a scalar $\mathcal{M} \in \{0, 1\}$ and is independently sampled from a Bernoulli distribution with a probability parameter p for each layer.

The scale vector multiplies the weighted sum of each layer. Therefore, setting the scale values to zero (similar to traditional Dropout) would lead to a complete loss of information in that layer. To address this problem, we introduce an alternative approach called *Unitary Dropout*. In this method, when the randomly generated Dropout mask is *zero*, all elements associated with the scale vector are set to *one*.

As a result, during forward propagation, the network ignores the scale factors that correspond to the Dropout mask being *zero*, while the scaling factor retains its original value when the randomly generated Dropout mask is set to *one*. Fig. 4.18 shows the scale Dropout concept during the train and inference time.

Although we have focused on *Unitary Dropout* in this paper due to their simple implementation in the CiM architecture, other alternatives can also be considered. For instance, *Average Scale Dropout* and *Random Scale Dropout*. In *Average Scale Dropout*, instead of setting the scale vector to *one*, it involves dropping to the average of the scale vector, $\bar{\boldsymbol{\alpha}} = \frac{1}{C_{out}} \sum_{i=1}^{C_{out}} \boldsymbol{\alpha}_i$. On the other hand, the Random Scale-Dropout method involves replacing the dropped scale with a random value sampled from a predefined distribution, for example, a uniform distribution.

Additionally, to reduce the number of Dropout modules to one per layer in the CiM architecture, the entire scale vector is dropped at the same time, which is referred to as "vector dropout." Furthermore, since each layer in an NN sequentially processes an input for inference, the scale dropout module can be shared across layers. Consequently, **only one Dropout module is required per layer**. Note that, if needed, the proposed scale Dropout can be applied to the scale vector element-wise at the cost of a large number of Dropout modules.

4.3.1.3. Co-adaptation Mitigation

The introduction of the proposed scale Dropout imposes randomness in the scale vector and, in turn, the activation of a layer. Thereby, it can potentially reduce co-adaptation between the scale vector and the binary weights. When $\boldsymbol{\alpha}$ is treated as a random variable during training, the model is less dependent on specific scale values, promoting a more diverse range of features in the BNN. This phenomenon can be expressed mathematically as increased variance in the learned representations across the network, thus reducing co-adaptation.

4.3.1.4. Choosing Dropout Probability

To choose a Dropout probability of the scale Dropout, we propose a *layer-dependent adaptive scale Dropout* method. Specifically, a Dropout probability of 10% or 20% is used on layers with a comparably smaller number of parameters, but a larger Dropout probability, e.g., 50%, is used on layers with a larger number of parameters. Consequently, unlike works [84, 86], where many different kinds of implementation with different locations for the Dropout layer need to be explored, our approach does not require such exploration, as the Scale-Dropout is applied to all the binary layers. Also, it is not necessary to explore the various Dropout rates. Consequently, our approach stands out as a more deployment-ready solution compared to related works.

4.3.1.5. Learning with Scale-Dropout

The proposed BayNN with *Scale-Dropout* can be trained using stochastic gradient descent, similar to standard BNN using existing algorithms such as [44, 45]. The only difference is that for each forward pass during training, we sample a random *scaled network* by applying Scale-Dropout. The forward and backward propagation for each iteration is performed only on this *scaled network*. The gradients for each parameter are averaged over the training instances of each mini-batch. The training objective combining a Bayesian approximation and *Scale-Dropout* is discussed in Section 4.3.2.

Although *Scale-Dropout* alone offers several benefits, using *Scale-Dropout* in conjunction with common regularization techniques such as L2 regularization, learning rate scheduling, data augmentation, and momentum for the gradient descent algorithm further improves accuracy.

4.3.2. Scale-Dropout as a Bayesian Approximation

To reiterate, an NN with standard Dropout can be used as an approximate method of Bayesian inference. Gal et al. [35] showed that learning an NN with Dropout and L2 regularization is equivalent to a Gaussian process. The optimization objective of their approach, named MC-Dropout, is given by

$$\mathcal{L}(\boldsymbol{\theta})_{\text{MC-Dropout}} = \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) + \lambda \sum_{l=1}^{L} (||\mathbf{W}^l||_2^2 + ||\mathbf{b}^l||_2^2). \tag{4.21}$$

In this paper, we propose *Monte Calo (MC)-Scale Dropout* based Bayesian approximation that uses *Scale-Dropout* in place of the standard Dropout for Bayesian inference. Our approach expands the MC-Dropout approaches [84, 35], for BNN and better efficiency with specific learning objectives. In the following section, the learning objective and how to obtain the model uncertainty for the *(MC)-Scale Dropout* are discussed in detail.

4.3.2.1. Learning Objective

For the proposed *Monte Calo (MC)-Scale Dropout* objective, we introduce a regularization function for the scales $\boldsymbol{\alpha}$. Specifically, we design a regularization function that encourages the scale factor to be positive to preserve the sign of the computed $\mathbf{z}^{(l)}$ of a layer l. Also, it encourages the scale factor to be centered around one, so it scales up or down the element of $\mathbf{z}$ based on their contribution to the loss.

To achieve a Bayesian approximation, we use a similar approach to MC-Dropout. However, in MC-Dropout, activations are dropped to zero, which inspires the L2 regularization to push the weights towards zero. On the contrary, in our *Unitary Dropout* approach, the scale factors are dropped to *one*. This promotes a regularization effect that encourages the scale vector to center around one, a key distinction that aligns better with the nature of binary networks where weights are binarized to -1 or 1. The regularization function can be mathematically described by

$$\varphi \sum_{l=1}^{L} (1 - \mu_{\boldsymbol{\alpha}}^l)^2. \tag{4.22}$$

Here, $\mu_{\boldsymbol{\alpha}}^l$ is the mean of the scale vector of a layer l and φ is the hyperparameter for controlling the strength of the regularization.

Despite the regularization of the scales, we also optionally apply the L2 regularization to the weights. Applying L2 regularization is a challenge in BNN. In BNN, real-valued proxy weights are binarized to +1 or -1, therefore, applying L2 regularization to either of them may not be beneficial [84].

However, L2 regularization can be implemented in the actual real-valued weights, with binarization applied to the normalized weights within the output channel dimensions [86]. Opting for channel-wise normalization also proves advantageous in reducing binarization errors [47]. To achieve this, the channel-wise mean is first computed:

$$\mu_c = \frac{1}{k_h \times K_w} \sum_{K_h} \sum_{K_w} \mathbf{W}. \tag{4.23}$$

Here, k_h and k_w represent the height and width of the kernels in the weight matrix, the last two dimensions of $\mathbf{W}$. Subsequently, the channel-wise mean μ_c is subtracted from the proxy weights (real-valued):

$$\hat{\mathbf{W}} = \mathbf{W} - \mu_c. \tag{4.24}$$

Following that, the channel-wise standard deviation is calculated on zero-centered weights.

$$\sigma_c^2 = \frac{1}{k_h \times k_w} \sum_{k_h} \sum_{k_w} \mathbf{W}^2 - \hat{\mathbf{W}}^2. \tag{4.25}$$

Note that σ_c^2 calculation is simplified for efficiency reasons. Lastly, the channel-wise standard deviation divides the zero centered weight for channel-wise normalization as:

$$\tilde{\mathbf{W}} = \frac{\hat{\mathbf{W}}}{\sigma_c}. \tag{4.26}$$

Consequently, binarization on the channel-wise normalized weights can be defined as:

$$\mathbf{W}^* = \begin{cases} +1 & \text{if } \tilde{\mathbf{W}} \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{4.27}$$

Note that channel-wise weight normalization has become standard practice in modern BNN models.

The overall objective of the MC-Scale Dropout with both scales and weight Dropout is defined as:

$$\mathcal{L}(\boldsymbol{\theta})_{\text{MC-Scale Dropout}} = \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) + \lambda \sum_{l=1}^{L} ||\mathbf{W}^l||_2^2 + \varphi \sum_{l=1}^{L} (1 - \mu_{\boldsymbol{\alpha}}^l)^2. \tag{4.28}$$

Here, λ is the weight decay hyperparameter of the weight regularization.

4.3.2.2. Obtaining Model Uncertainty

To obtain the uncertainty of the model, we perform T forward passes with the proposed *Scale-Dropout* enabled during Bayesian inference. During each of the T forward passes, we sample an independent and identically distributed random Dropout mask from the Bernoulli distribution for each layer $\{\mathcal{M}_t^{(l)}, \cdots, \mathcal{M}_t^{(L)}\}_{t=1}^T$, giving T stochastic scale vectors $\{\hat{\boldsymbol{\alpha}}_t^{(l)}, \cdots, \hat{\boldsymbol{\alpha}}_t^{(L)}\}_{t=1}^T$ and ultimately stochastic weighted sums $\{\mathbf{z}_t^{(l)}, \cdots, \mathbf{z}_t^{(L)}\}_{t=1}^T$. The predictive mean is given by:

$$E_{q(y^*|x^*,\mathcal{D})}(y^*) \approx \frac{1}{T} \sum_{t=1}^{T} \tilde{y}_t^*(x^*, \mathbf{z}_t^{(l)}, \cdots, \mathbf{z}_t^{(L)}) \tag{4.29}$$

Here, x^* is the inferece input, $q(y^* \mid x^*, \mathcal{D})$ is the posterior distribution, $\tilde{y}$ is the stochastic prediction, and y^* is the final prediction. We refer to this Monte Carlo estimate as the *MC-Scale Dropout*. In practice, this is equivalent to performing T stochastic forward passes through the network and averaging the results. In the literature, this is known as model averaging [35].

In terms of the posterior distribution of the output, equations 4.16 can be modified as

$$\mathbf{z}^{(l)} = S^{(l)} \odot \text{diag}(d) \tag{4.30}$$

$$d^{(l)} \sim \text{Bernoulli}(p) \text{ for } l = 1, \cdots, L \tag{4.31}$$

Here, S represents the weighted sum of a layer. Batch normalization is applied to $\mathbf{z}^{(l)}$. Thus, the sampling process of the Dropout mask is the same as that of the MC-Dropout. In an empirical evaluation (shown later in the 4.3.4.6), we observed that the distribution of the output for each class approaches a Gaussian distribution as the number of stochastic forward passes (T) through the network increases. This is due to the aggregate effect of the scalar dropout mask over many forward passes, which can be seen as introducing a form of multiplicative noise.

Uncertainty estimates of the prediction can be obtained from the variance of the T forward passes as

$$\mathrm{Var}_{q(y^*|x^*,\mathcal{D})}(y^*) \approx \frac{1}{T}\sum_{t=1}^{T}(\tilde{y}_t^*(x^*, \mathbf{z}_t^{(l)}, \cdots, \mathbf{z}_t^{(L)}) - E_{q(y^*|x^*,\mathcal{D})}(y^*))^2 \quad (4.32)$$

In addition, the $\mathcal{K}\%$ confidence interval (CI) can also be used as an uncertainty estimate of the MC-Scale Dropout model. According to the central limit Theorem, for sufficiently large T, $\{\tilde{y}_1^* \cdots \tilde{y}_T^*\}$ follow a normal distribution. For a $K\%$ confidence interval, we use the percentiles of the predictions. Let $\mathcal{G}_{\vartheta/2}$ be the $\vartheta/2$-th quantile of the predictions. Where $\vartheta = 1 - K/100$. Consequently, the $K\%$ confidence interval is given by

$$\mathrm{CI} = \left[\mu_y - \mathcal{G}_{\vartheta/2}\frac{\sigma_y}{\sqrt{T}}, \mu_y + \mathcal{G}_{\vartheta/2}\frac{\sigma_y}{\sqrt{T}}\right]. \quad (4.33)$$

Here, μ_Y and σ_Y represent predictive mean $E_{q(y^*|x^*,\mathcal{D})}(y^*)$, and variance $\mathrm{Var}_{q(y^*|x^*,\mathcal{D})}(y^*)$ from formulas 4.29 and 4.32, respectively. For sufficiently large T, the confidence interval can be approximated by directly calculating the $\frac{100-k}{2}$ and $\frac{100+k}{2}$ quantile (for a $K\%$ CI) of the predictions as

$$\mathrm{CI} \approx \left[\mathrm{percentile}\left(\frac{100-k}{2}\right), \mathrm{percentile}\left(\frac{100+k}{2}\right)\right]. \quad (4.34)$$

4.3.3. Hardware Implementation

4.3.3.1. Modelling Spintronic-based Scale Dropout

In our design, only *one* spintronic-based Dropout (namd here Spin-ScaleDrop) module is designed and implemented for the entire neural network. Thus, the proposed Spin-ScaleDrop module is reused for all layers of the CiM architecture. After the computation of a layer is performed, a new Dropout mask from the Spin-ScaleDrop Module module is sampled for the next layer.

However, due to the manufacturing and infield variation of the MTJs in the Spin-ScaleDrop, the Dropout probability itself becomes a stochastic variable. Similar to our previous works, we model the Spintronic-implemented dropout probability in CiM as a Gaussian distribution, the mean of the distribution μ represents the expected Dropout probability, and σ represents the device variations. Therefore, the probability of Dropout p of a layer l can be modeled as

$$\hat{p}_l = p_l + \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}(\mu, \sigma^2). \quad (4.35)$$

Here, $\hat{p}_l$ denotes the probability of Dropout with process variation. The feed-forward operation expressed in equation 4.16 remains the same, with only $\hat{p}_l$ used as the Dropout probability. Note that a probability has to be in $[0, 1]$, as p is usually chosen between 0.1 and 0.5, and it is unlikely that p crosses this range due to variation. The Dropout probability typically varies from 3% to 10% from the trained one.

4.3.3.2. Designing Spintronic-Based Scale Dropout Module

The Spin-ScaleDrop module is designed by harnessing the stochastic regime of an MTJ and is utilized as a random number generator. The probability density function governing the switching of the SOT-MTJ follows an exponential distribution and is expressed as [180]:

$$p_{\mathrm{sw}} = 1 - \exp\left(\frac{t}{\tau}\right) \quad (4.36a)$$

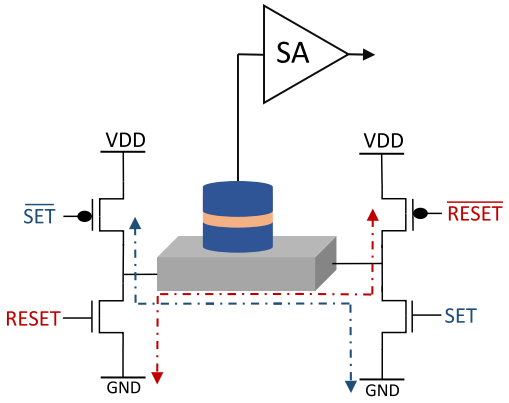


Figure 4.19.: Spin Scale-Dropout Module based on SOT MTJ.

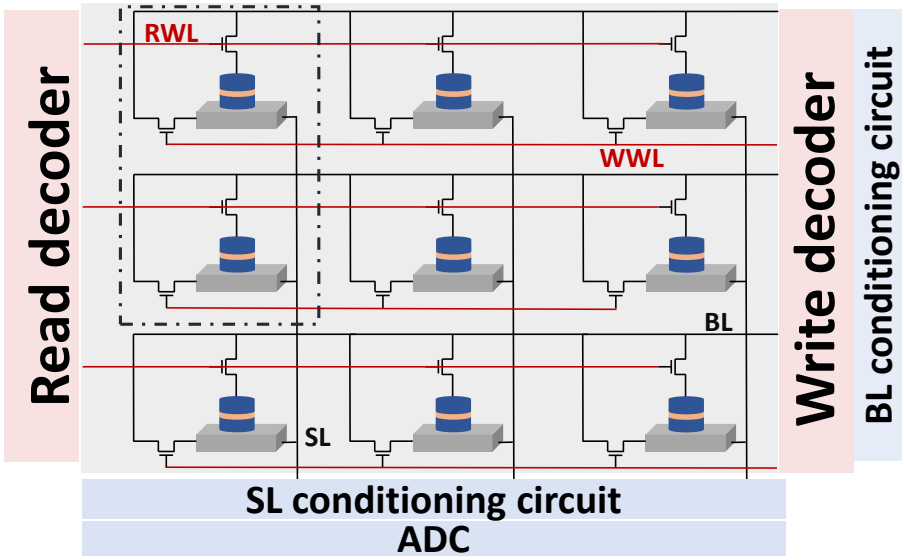


Figure 4.20.: Binary SOT crossbar array for the Bayesian inference.

$$\tau = \tau_0 \exp\left[\frac{\Delta}{k_B \mathcal{T}}\left(1 - 2\frac{I}{I_{c0}}\left(\frac{\pi}{2} - \frac{I}{I_{c0}}\right)\right)\right] \tag{4.36b}$$

Here, Δ is the thermal stability factor, I is the applied current through the SOT-track, t is the pulse duration, τ_0 is the attempt time, I_{c0} is the critical current at 0 K, k_B is the Boltzmann constant and $\mathcal{T}$ is the temperature. I_{c0} represents the minimum current required to switch the MTJ. The equation equation 4.36 is used to model the switching behavior of the SOT-MTJ for different switching currents while keeping the pulse width fixed at 10 ns. To generate the bidirectional current across the SOT track, four transistors are added, as shown in Fig. 4.19. The desired switching probability of, for example, 50%, is achieved by programming the MTJs through successive "SET" and "RESET" operations.

To ensure reliable MTJ switching, the write duration is set to 10 ns for the SET operation and to 5 ns for the RESET operation. The state of the MTJ is read using a Sense Amplifier (SA, in Fig.4.19). The SET and RESET cycles are repeated to generate a stochastic sequence. The Scale Dropout Module allows for the stochastic activation of the Scale vector that is stored in the neighboring memory.

4.3.3.3. Proposed Spintronics-based CiM Architecture

In spintronic-based CiM architectures, the SOT-MRAM devices are arranged in a crossbar fashion, with an MRAM device at each crosspoint (see Fig. 4.20). For inference, the mapping of the trained binary weights to

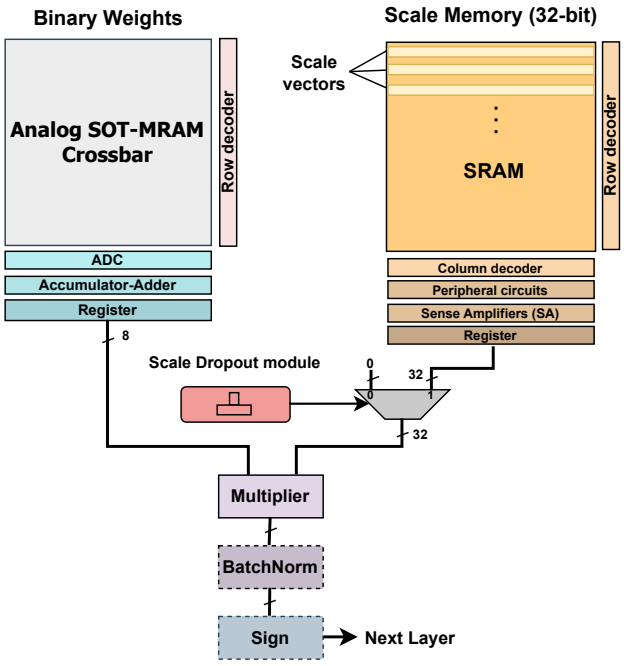


Figure 4.21.: Proposed inference architecture for Scale-Dropout.

the array is performed with a one-time write operation. To reiterate, in BNN, XNOR and the bit-counting operation are performed instead of the weighted sum operation [44]. The XNOR operation in CiM is shown in Table 2.1 and the encoding of the respective +1 and −1 weights with the complementary bit cell is shown in Fig. 4.20.

The mapping of the weight matrix of various NN layers to the crossbar arrays are done using the mapping strategy employed as discussed in Section 2.9.1.

During Bayesian inference (online operation), each element of the binary input vector x for a layer is converted into a (0, 1) or (1, 0) signal and fed into the crossbar array for inference. This architecture allows for parallel computation and outputs the weighted sum results as currents flow through each source line. Finally, the analog currents are converted to digital signals using Analog-to-Digital Converters (ADCs) and passed on to the Accumulator-Adder module to sum up the partial matrix-vectors multiplication. These partial multiplications are then stored in registers and multiplied with the Scale memory.

Regarding the scale vectors, they are stored in a nearby 32-bit SRAM memory. In the scale memory, each row stores a scale vector of a layer. The column dimension of the SRAM memory depends on the maximum number of neurons or channels within the NN layers, and the row dimension depends on the number of layers in the model. This scale vector is subsequently applied, depending on the stochastic activation by the Scale Dropout module, using a multiplexer.

Recent state-of-the-art CNN topologies, e.g., ResNet and DenseNet, use skip connections. In CiM architectures, skip-connection can be implemented by selectively routing the output signals through the crossbars and summing them with digital circuits. Since layer-by-layer computations are sequential, signals for these connections can be stored in a buffer memory until the computation of the following layers is completed.

4.3.4. Evaluation

4.3.4.1. Evaluation Setup

In distribution (ID) Dataset To evaluate both predictive performance and uncertainty estimation, we have used several challenging benchmark and real-world biomedical in-distribution datasets on various

learning paradigms (classification and semantic segmentation) in the context of Bayesian deep learning. An in-distribution dataset refers to a set of data samples that come from the same distribution as the data the model was trained on. For example, if a model is trained on images of an aircraft, automobile, bird, cat, deer, dog, frog, horse, ship, and truck from CIFAR-10, then, during inference, more images of CIFAR-10 (although not seen during training) would be considered an in-distribution dataset.

Specifically, for classification, we have used the CIFAR-10. Furthermore, for biomedical semantic segmentation, breast ultrasound scans (for breast cancer)[162], COVID-19 lung computed tomography (CT) [181], and skin cancer [182]. The breast cancer dataset containing ultrasound scans is a vital resource used for the early detection of breast cancer, one of the leading causes of death among women worldwide. The dataset is classified into three classes, normal, benign, and malignant images, and has a total of 780 images with a size of 500×500 pixels on average. On the other hand, the Skin Cancer dataset for Biomedical Segmentation contains 200 dermoscopic images of shape 572×765 pixels with their corresponding label masks. Accurate prediction of skin cancer allows computer-aided diagnostic systems to assist medical professionals in the early detection and precise delineation of skin lesions. Lastly, the COVID-19 lung CT dataset contains anonymized human lung CT scans with different levels of severity in COVID disease.

Evaluating the proposed method on various datasets shows its scalability and generality. Note that semantic segmentation, which involves segmenting an image into multiple sections and labeling each pixel with its corresponding class label, is regarded as more difficult than classification tasks due to its finer granularity.

To improve accuracy, we applied random data augmentation and dataset normalization to all the datasets during training. For example, for CIFAR-10 datasets, we have applied RandomHorizontalFlip and RandomResizedCrop type random data augmentation.

OOD Dataset We used six additional OOD datasets to evaluate the efficacy of our method in dealing with data uncertainty: 1. Gaussian noise ($\hat{\mathcal{D}}_1$): Each pixel of the image is generated by sampling random noise from a unit Gaussian distribution, $\mathbf{x} \sim \mathcal{N}(0, 1)$, 2. Uniform noise ($\hat{\mathcal{D}}_2$): Each pixel of the image is generated by sampling random noise from a uniform distribution, $\mathbf{x} \sim \mathcal{U}(0, 1)$, 3. CIFAR-10 with Gaussian noise ($\hat{\mathcal{D}}_3$): Each pixel of the CIFAR-10 images is corrupted with Gaussian noise, 4. CIFAR-10 with uniform noise ($\hat{\mathcal{D}}_4$): Each pixel of the CIFAR-10 images is corrupted with uniform noise, 5. SVHN: Google street view house numbers dataset [183], and 6. STL10: a dataset containing images from the popular ImageNet dataset [184]. Each of these OOD datasets contains 8000 images, and the images have the same dimensions as the original CIFAR-10 dataset (32×32 pixels).

Evaluated Topologies and Training setting The proposed Scale-Dropout is evaluated for its predictive performance and uncertainty estimation in state-of-the-art convolutional NN (CNN) topologies, including ResNet [9], and VGG [185] for benchmark classification tasks. In the case of biomedical image segmentation tasks, U-Net [164], and Bayesian SegNet [165], topologies are used. The U-Net topology consists of a contracting path and an expansive path with skip connections, which gives it the U-shaped architecture. On the other hand, Bayesian SegNet is a deep convolutional encoder-decoder architecture for semantic image segmentation.

All models are trained with the Adam optimization algorithm with default settings in the PyTorch framework to minimize the proposed objective function with a weight decay rate of $\lambda = 1 \times 10^{-5}$ and $\varphi = 1 \times 10^{-5}$. Classification and segmentation tasks are trained for 300 epochs.

All weights and activations of models for classification tasks are binarized (1-bit model). The activations of biomedical semantic segmentation models are quantized to 4 bits, but their weights are kept binary. As stated previously, semantic segmentation tasks are more difficult, and therefore, they require slightly more bit precision at the activation for accurate predictions. We have used the activation quantization algorithm proposed in [173] to quantize the activations to 4 bits. Since 1-bit weights are still maintained, the crossbar structure does not need to be modified. In fact, only peripheral modifications, such as ADC with increased bit resolution, are required.

Table 4.14.: Energy estimation for the different elements of the architecture for one reading operation.

Circuit	Energy	Circuit	Energy
Memory (Decoding/Sensing)	4.76 pJ	Adder-Accumulator	0.12 pJ
Spintronic RNG	3.80 pJ	Comparator	0.01 pJ
Averaging block	18.42 pJ	Crossbar array	0.65 pJ

We have used the recently proposed IrNet [47] binarization algorithm to implement the proposed learnable scale and Scale-Dropout. Note that (to our knowledge) any binarization algorithm can be extended with our method with slight modification, i.e., add a learnable scale vector and scale Dropout.

Evaluation Metrics For segmentation tasks, the same metrics as in our SpinDrop work [85] are used. Specifically, pixel-wise accuracy, intersection-over-union (IoU), sensitivity, specificity, area under the ROC curve (AUC), F1 score, and precision. On the other hand, classification tasks are evaluated for their inference accuracy.

(Epistemic) Uncertainty estimation of the models is evaluated on predictive variations, entropy, and confidence interval with $K = 95\%$ based on Equations 4.32 and 4.34, respectively. Out-of-distribution data is detected similar to our grouped Dropout work [86]. Specifically, the classification as ID or OOD depends on whether the maximum value from the 10th percentile of the outputs is less than the 0.95 SoftMax score (for OOD) or not (for ID). The underlying idea of our OOD detection is based on our SpinDrop work [85]. That is, for in-distribution data, most confidence scores of the T MC runs are high and close to one another, resulting in low variance. In contrast, for out-of-distribution data, confidence scores exhibit higher variance.

Architectural Simulation To carry out the architectural simulation, we first obtained the circuit specifications for the peripheral blocks, as outlined in Section 4.3.3. We then independently simulated each component of the architecture to gauge its energy utilization. Both the crossbar array and the Spin Scale-Dropout module were analyzed using an electrical simulator, such as the Simulation Program for Integrated Circuit (SPICE), to assess their energy consumption. The use of high-resistance SOT devices [186], in conjunction with the binary nature of the network, serves to reduce the overhead related to peripheral elements.

The Accumulator-Adder, Comparator, and Averaging circuits were synthesized using the Synopsys Design Compiler, leveraging the TSMC 40 nm low-power Process Design Kit (PDK). For the CiM operation, decoding and sensing were assessed at the circuit-array level using NVsim (NonVolatile memory simulator) [160]. To achieve this, we modified the NVsim simulator to accommodate multiple active cells, thus simulating CiM operation accurately. Additionally, we substituted the single-bit sense amplifiers with multi-bit ADCs. Performance metrics for each discrete component are shown in Table 4.14.

4.3.4.2. Evaluation

4.3.4.3. Predictive Performance

Comparison With State-of-the-Art Algorithms The predictive performance of our method is comparable to the SOTA binary Bayesian NN methods, as shown in Table 4.15 on a range of CNN architectures, including VGG, ResNet-18, and ResNet-20, evaluated on the CIFAR-10 dataset. In the worst case, the predictive performance is 1.45% below the SpinDrop [84] method for the VGG topology. Here, we assumed that there are no device variations in the spintronics-based scale Dropout module. For a fair comparison, we used the same network size as those used in their work. However, the hardware implementation of our solution may lead to a smaller area and a better power-performance product owing to a simpler spintronics-based Dropout module design. In our analysis, we have used layer-dependent adaptive Dropout rates (See

Table 4.15.: Predictive performance of the proposed MC-SparialDropout method in comparison with SOTA methods on CIFAR-10. The accuracy closest to the MC-Dropout is in bold, and the number in the bracket shows the standard deviations of the accuracy after different repetitions.

Topology	Method	Bit-width (W/A)	Bayesian	Inference Accuracy
ResNet-18	FP	32/32	No	93.0%
	RAD [175]	1/1	No	90.5%
	IR-Net [47]	1/1	No	91.5%
	MC-Dropout(HTanh) [35]	32/32	Yes	90.56%
	SpinDrop [84]	1/1	Yes	90.48%
	Spatial-SpinDrop [86]	1/1	Yes	91.34%
	Proposed	**1/1**	**Yes**	**91.52%(±0.047)**
ResNet-20	FP	32/32	No	91.7%
	DoReFa [176]	1/1	No	79.3%
	DSQ [177]	1/1	No	84.1%
	IR-Net [47]	1/1	No	85.4%
	MC-Dropout(HTanh) [35]	32/32	Yes	86.94%
	Spatial-SpinDrop [86]	1/1	Yes	84.71%
	Proposed	**1/1**	**Yes**	**86.04%(±0.039)**
VGG	FP	32/32	No	91.7%
	LAB [178]	1/1	No	87.7%
	XNOR [45]	1/1	No	89.8%
	BNN [44]	1/1	No	89.9%
	RAD [175]	1/1	No	90.0%
	IR-Net [47]	1/1	No	90.4%
	MC-Dropout (HTanH) [35]	32/32	Yes	89.49%
	MC-Dropout (ReLU) [35]	32/32	Yes	91.64%
	MC-DropConnect [36]	32/32	Yes	91.36%
	SpinDrop [84]	**1/1**	**Yes**	**91.95%**
	Spatial-SpinDrop [86]	1/1	Yes	90.34%
	Proposed	1/1	Yes	90.45%(±0.052)

Section 4.3.1.4) for scale Dropout. The low variance in inference accuracy (numbers in parentheses) shows the stability of the proposed approach.

In terms of the activation function, the proposed binary BayNN uses the $\mathrm{Sig} \cdot (.)$ function, which is an approximation of the hard Tanh function. In this case, the proposed method performs similarly to the MC-Dropout method. However, in the case of the ReLU activation function in the MC-Dropout model, the accuracy of the MC-Dropout model increases. Thus, the difference between the proposed method and the MC-Dropout increases to about $\sim 1\%$.

Furthermore, our proposed method improves inference accuracy by up to 6.74 compared to the SOTA point estimate BNN algorithm. However, since our method is built on top of the IR-Net BNN algorithm [47], predictive performance should be comparable to their approach. As depicted in Table 4.15, the predictive performance is, in the worst case, 0.18% lower, which is negligible. Similarly, accuracy is comparable to the full precision model, depicting that our method, in general, does not increase quantization error. Note that in the full precision model, the ReLU function is used as the activation for the convolution layers, while our proposed method uses the activation function $\mathrm{sign}(x)$ for all layers where activations are applied.

For biomedical image segmentation tasks, the proposed method outperforms the full-precision MC-Dropout method by up to 6.4% in terms of IoU score. In the worst scenario, our method results in a 69.69% reduction in the IoU score for the breast cancer dataset. Additionally, our approach outperforms the MC-Dropout method in most other metrics. Table 4.17 presents a summary of the results.

Table 4.16.: Evaluation of the inference accuracy of the proposed Spintronics-based Scale-Dropout with variations in the Dropout module. Variations in the probability of Dropout p increased from 1× to 3×, and the baseline model is the ideal scenario without variation.

Topologie	Baseline	Trained w/ Variations			Trained w/o Variations		
		Var. 1×	Var. 2×	Var. 3×	Var. 1×	Var. 2×	Var. 3×
ResNet-18	91.52%	91.78%	91.77%	91.71%	91.65%	91.59%	91.58%
VGG	90.45%	90.52%	90.52%	90.55%	90.28%	90.26%	90.30%

The predictive performance is qualitatively shown in Figure 4.25 for each dataset (with two examples). The sixth and third columns show the prediction mask for MC-Dropout and our method, respectively. It can be observed that the segmentation masks for the proposed method are similar to MC-Dropout and ground truth. In general, misclassified pixels are around the boundary of ground-truth masks.

Impact of MC Runs on Inference Accuracy We observed that using Monte Carlo sampling (T forward pass) for Bayesian inference generally enhances predictive performance across all datasets. For example, the inference accuracy of the ResNet-20 model increases from 84.63% to 86.05%. In our evaluation, twenty samples (T = 20) for the larger model and fifty samples (T = 50) for the smaller model were used for Bayesian inference. Fig. 4.24 (a) shows that the proposed method requires a smaller number of samples, with inference accuracy plateaus around T = 20 to 50. In comparison, MC-Dropout and MC-DropConnect methods require 100, and 90 Monte Carlo sampling, respectively, to achieve the maximum inference accuracy, as reported in [36] for CIFAR-10. In our experiment (see Fig. 4.24 (b)), we observed that the MC-DropConnect method plateaus at 100 Monte Carlo runs, and the MC-Dropout method plateaus at 200 Monte Carlo runs on the same model (VGG) and dataset. Therefore, *our method requires up to* 180 *less Monte Carlo sampling*, leading to 10× less XNOR and bit-counting operation, energy consumption, and latency for *each Bayesian inference result*. For a fair comparison, we assume the same NN topology, hardware architecture, and memory device technology. However, it should be noted that T at which accuracy plateaus can vary from task to task and from model to model.

Performance of BayBNN with Spin-ScaleDrop We have shown that the predictive performance of our method is comparable to that of the full precision and binary implementations, assuming that the Spintronics Dropout module remains unchanged. However, it should also be tolerant to manufacturing and thermal variations in the Spintronic-based Dropout module.

To this end, we performed a small ablation study on the CIFAR-10 dataset with ResNet-18 and VGG topologies with models trained with and without variations in the Dropout module. Specifically, in one study, we trained both models with no variation in the probability of Dropout, but during Bayesian inference, we evaluated the model against our proposed Spintronic-based Dropout with up to 3× the standard deviation σ of the manufacturing variations. This means that the Dropout probability of each neuron can fluctuate by ±10% from the trained value. In this case, a slight improvement (+0.13%) in predictive performance is observed for the ResNet-18 model, but for the VGG model, a slight reduction in inference accuracy (−0.19%) is observed. Nevertheless, the inference accuracy for both models remains close to the baseline accuracy. Furthermore, increasing the variation of a model from 0× to 3× has a negligible effect on the inference accuracy.

In the other case, the NN is trained considering the variation in the Dropout module (see Section 4.3.3.1). In this case, unlike in the previous case, there is a slight improvement in predictive performance (up to +0.26%) for both models compared to the baseline. Variation in the Dropout probability leads to more stochastically during Bayesian inference, and as a result, accuracy improves slightly. The results are summarized in Table 4.16.

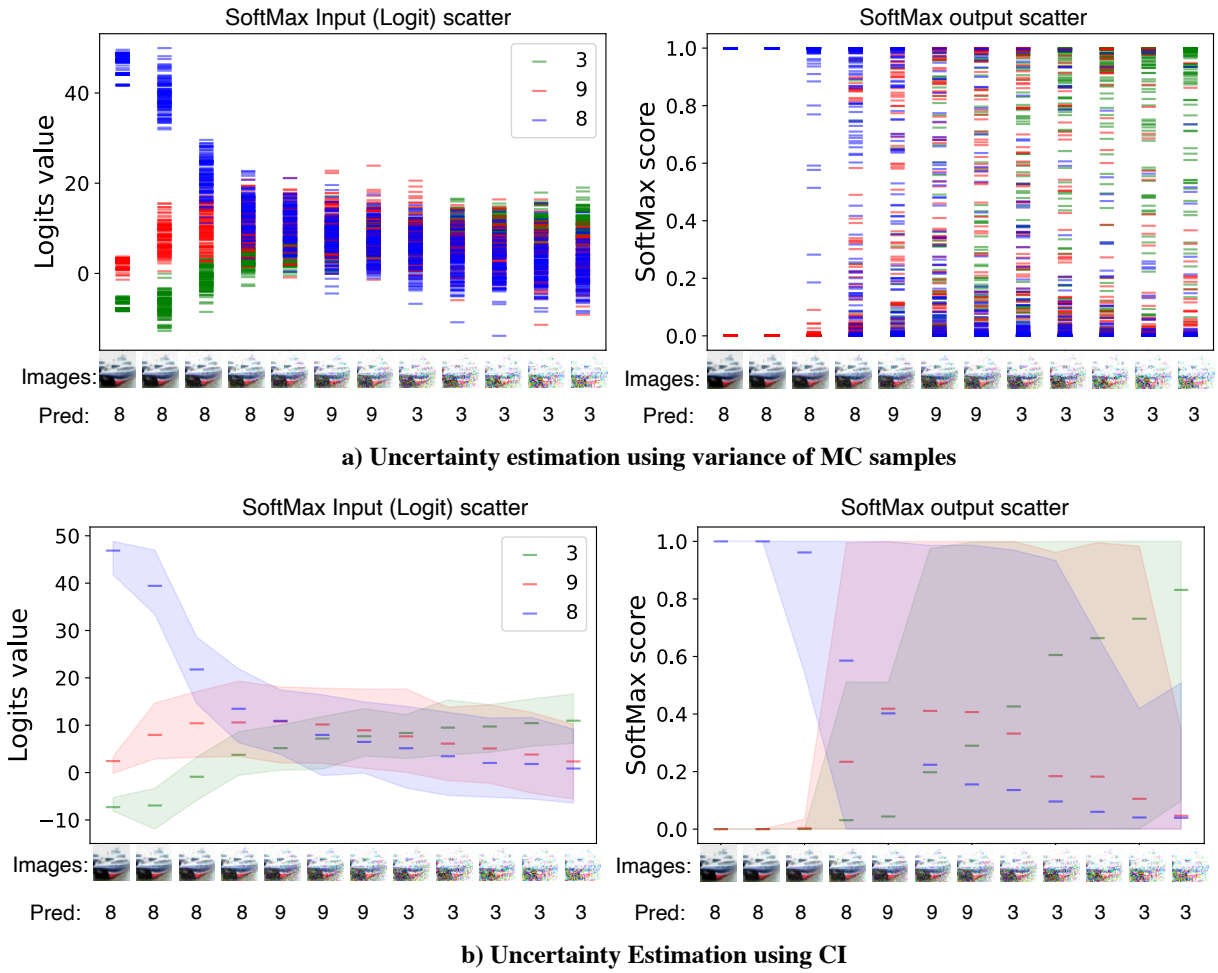


Figure 4.22.: Detecting Distribution Shift on CIFAR-10: a) A scatter and b) 95% confidence interval of 100 forward passes of the softmax input (logits) and output for Scale-Dropout VGG topology. Uniform noise of increasing strength is added to a randomly sampled image of a ship (leveled as 8). The uncertainty of the prediction increases with the data distribution shift, as shown by the high SoftMax scatter and the confidence interval. Although the model uncertainty is extremely high (best observed in color), the input for images 5 through 12 is classified as either a truck (leveled as 9) or a bird (leveled as 3). **It is recommended to view this figure in color.**

4.3.4.4. Uncertainty Estimation

Detecting Distribution Shift To show the effectiveness of the proposed Scale-Dropout method in detecting distribution shifts in the data, we conducted two experiments. In one experiment, we continuously added random noise from a uniform distribution to the input data with increasing strength. As shown in Fig. 4.22, the variance and confidence interval in the model logits (SoftMax input) and the predicted probability of the output classes (SoftMax output) increases as the strength of noise increases. In other words, the uncertainty in the prediction increases as the distribution dataset shifts away from the original distribution. However, despite the high uncertainty, the model predicts a truck or a bird.

On the other hand, we have performed another experiment with all images of the CIFAR-10 dataset on the VGG model continuously rotated up to 90°. It can be seen in Fig. 4.23 that as the images are rotated, the inference accuracy decreases, and the predictive entropy increases from the starting entropy. Our method is compared with deterministic as well as common uncertainty estimation techniques, namely MC-Dropout [35] and Deep Ensemble [187] with five randomly initialized models. The trend of decrease in inference accuracy is similar for all models. However, the Deep Ensemble slightly outperforms the proposed and other methods in terms of accuracy. Regardless, our proposed MC Scale-Dropout method produces significantly more predictive entropy compared to other methods, including the Deep Ensemble

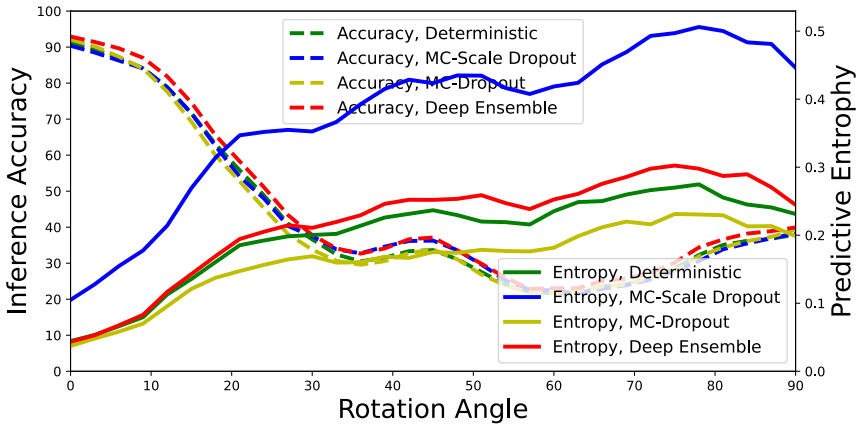


Figure 4.23.: The effect of distribution shift of inference images on inference accuracy (left y-axis) and predictive entropy (uncertainty estimate on right y-axis). Images are continuously rotated to introduce distribution shifts. The inference accuracy of all methods is reduced with the same trend, and the uncertainty of prediction increases with the data distribution shift. The uncertainty estimates of the proposed method outperform those of other methods, but the accuracy of the Ensemble method is higher in comparison. **It is recommended to view this figure in color.**

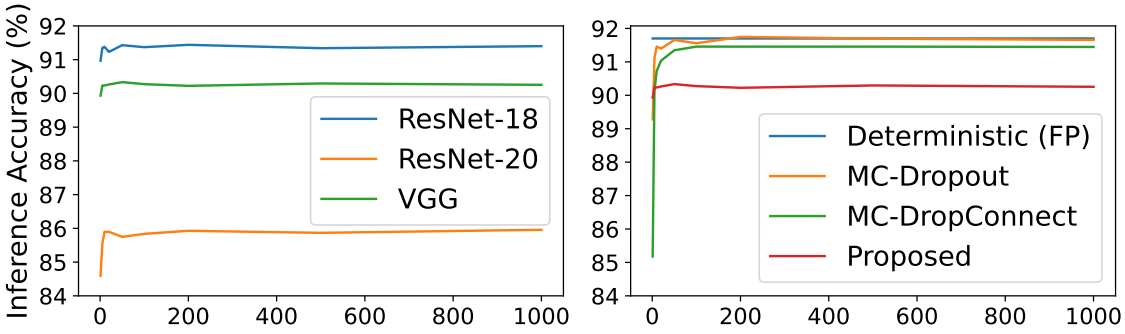


Figure 4.24.: Evaluation of the effect of Monte Carlo runs on the inference accuracy of the CIFAR-10 dataset on various topologies. **It is recommended to view this figure in color.**

method. This is because the proposed MC Scale-Dropout method effectively turns a single model into numerous ensembles by enabling Dropout during inference, allowing it to generate multiple predictions from a single model. Also, the proposed Scale Dropout can be interpreted as the addition of learnable multiplicative noise to the weighted sum of each binary layer. Whereas, the Deep Ensemble has limited models in the ensemble, e.g., five models. However, we believe that adversarial training with the deep ensemble may improve its uncertainty estimates. Consequently, our method can produce better uncertainty estimates compared to related works, even with a 1-bit model in this experiment.

Table 4.17.: The analysis of the proposed Scale-Dropout BayBNN on three biomedical segmentation datasets on SOTA topologies. A Dropout probability of 20% is used with 20 Monte Carlo samples for Bayesian inference. The best-performing matrices are in bold.

Topology	Dataset	Method	Bit-width (W/A)	Pixel-Acc	IoU	AUC	F1	Sensitivity	Specificity	Precision
U-Net	Skin Cancer	FP	32/32	95.10%	82.55%	98.81%	90.44%	89.04%	97.23%	91.88%
		MC-Dropout [35]	32/32	95.05%	81.67%	98.86%	89.91%	84.74%	**98.67%**	**95.74%**
		Proposed	1/4	**96.75%**	**88.07%**	**99.4%**	**93.66%**	**92.31%**	98.31%	95.04%
Bayesian SegNet	Breast Cancer	FP	32/32	97.47%	67.65%	96.60%	80.71%	76.95%	98.99%	84.84%
		MC-Dropout [35]	32/32	**97.83%**	**71.82%**	96.40%	**83.21%**	**78.23%**	**99.28%**	**88.86%**
		Spatial-SpinDrop [86]	1/4	96.32%	56.12%	94.31%	71.90%	68.50%	98.37%	75.64%
		SpinDrop [84, 85]	1/4	97.47%	68.17%	95.48%	80.13%	74.25%	99.18%	87.02%
		Proposed	1/4	97.69%	69.69%	**96.69%**	**82.14%**	77.19%	99.21%	87.76%
Bayesian SegNet	COVID-19 Lung CT	FP	32/32	99.53%	72.86%	99.11%	84.30%	83.17%	**99.78%**	**85.46%**
		MC-Dropout [35]	32/32	99.51%	72.88%	99.39%	84.32%	**86.99%**	99.7%	81.8%
		Proposed	1/4	**99.55%**	**74.43%**	**99.69%**	**85.34%**	85.84%	99.76%	84.85%

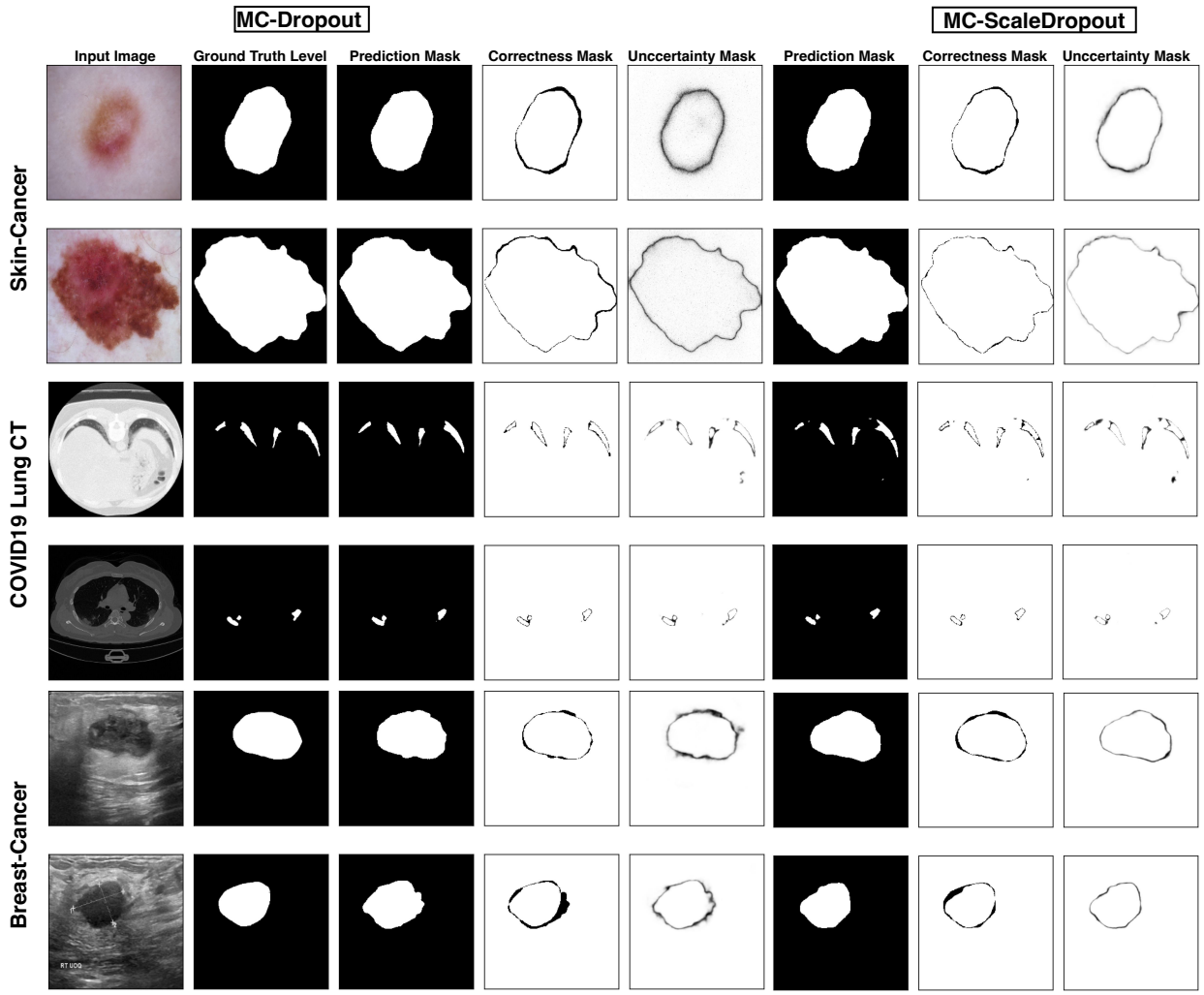


Figure 4.25.: The outcomes of semantic segmentation and uncertainty estimations for the Skin-Cancer, COVID-19 Lung-CT, and Breast Cancer datasets. Each row comprises the input image followed by the ground truth, prediction mask, correctness mask, and uncertainty mask for both the MC-Dropout and proposed MC-Scale Dropout methods. The correctness mask is a binary representation of accurate and inaccurate predictions. The uncertainty mask is the normalized [0, 1] uncertainty mask derived from twenty Monte Carlo samples. For prediction masks, a 0.5 threshold is used. On the correctness and uncertainty masks, the correct and certain regions are depicted in white. Similarly, a region that is incorrect or uncertain is displayed in black. **It is recommended to view this figure in color.**

Note that our scale Dropout method treats all the layers as probabilistic, but in MC-Dropout, only a few are created as such. As shown previously in our grouped Dropout [86] work, treating more layers as probabilistic can improve uncertainty estimates. However, in the MC-Dropout model, Dropout is applied to the extracted features from the convolutional layers to achieve similar inference accuracy. If Dropout is applied to all layers, the predictive entropy increases, but inference accuracy decreases significantly, e.g., by more than 3%. Our approach provides a good balance between uncertainty estimates without any noticeable degradation in accuracy.

Detecting Out-of-distribution Data We show that the model uncertainty increases as the distribution of the data shifts from the original distribution. Here, we perform an ablation study with six (definitive) out-of-distribution datasets.

As depicted in Table 4.18, our proposed method can achieve a detection rate of OOD of up to 100% across various model architectures and six different OOD datasets ($\hat{\mathcal{D}}_1$ through $\hat{\mathcal{D}}_6$). There are some variations in OOD detection rates across different architectures for the same OOD dataset. However, even in these cases, our method can consistently achieve a high OOD detection rate, with the lowest detection rate being 77.77%

Table 4.18.: Evaluation of the proposed MC-Scale Dropout method in detecting OOD across various topologies. All the models are trained on the CIFAR-10 dataset.

Topology	Method	$\hat{\mathcal{D}}_1$	$\hat{\mathcal{D}}_2$	$\hat{\mathcal{D}}_3$	$\hat{\mathcal{D}}_4$	$\hat{\mathcal{D}}_5$	$\hat{\mathcal{D}}_6$
VGG	**Proposed**	95.36%	95.40%	**96.19%**	**88.73%**	**98.02%**	85.24%
	Spatial-SpinDrop [86]	**99.99%**	**100%**	92.9%	78.91%	99.81%	**100%**
	SpinBayes [104]	99.86%	–	94.35%	–	80.33%	89.31%
ResNet-18	**Proposed**	**100%**	**100%**	97.49%	77.77%	91.53%	78.61%
	Spatial-SpinDrop [86]	100%	100%	**100%**	**92.26%**	**99.98%**	**97.39%**
ResNet-20	**Proposed**	96.51%	**100%**	90.34%	**93.55%**	**100%**	99.8%
	Spatial-SpinDrop [86]	**97.2%**	100%	**90.79%**	87.94%	99.03%	**99.81%**

Table 4.19.: Layer-wise and topology-wise overhead analysis of the proposed method in comparison to existing works SpinDrop [84] and Spatial-SpinDrop [86].

Layer-wise application of scale Dropout					
Method	Mapping Strategy	# of Dropout Modules	Area	Power Consumption	Sampling Latency
SpinDrop [84]	①	$K*K*C_{in}$	$79833.6\mu m^2$	$51.84mW$	$15ns$
	②	$K*K*C_{in}$	$79833.6\mu m^2$	$51.84mW$	$15ns$
Spatial-SpinDrop [86]	①	C_{in}	$8870.4\mu m^2$	$5.76mW$	$15ns$
	②	C_{in}	$8870.4\mu m^2$	$5.76mW$	$15ns$
Proposed	①	**1**	$\mathbf{34.65\mu m^2}$	**0.0225 mW**	$15ns$
	②	**1**	$\mathbf{34.65\mu m^2}$	**0.0225 mW**	$15ns$
Topology-wise application of scale Dropout					
Method	Adaptive Avg. Pool	# of Dropout Modules	Area	Power Consumption	Sampling Latency
SpinDrop [84]	Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	$15ns$
	Not Used	$K*K*C_{out}$	$159667.2\mu m^2$	$103.68mW$	$15ns$
Spatial-SpinDrop [86]	Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	$15ns$
	Not Used	C_{out}	$17740.8\mu m^2$	$11.52mW$	$15ns$
Proposed	Used	**1**	$\mathbf{34.65\mu m^2}$	**0.0225 mW**	$15ns$
	Not Used	**1**	$\mathbf{34.65\mu m^2}$	**0.0225 mW**	$15ns$

on the ResNet-18 model with $\hat{\mathcal{D}}_4$ dataset. However, when the threshold for SoftMax confidence increases from 95% to 99%, the OOD detection rate in the dataset $\hat{\mathcal{D}}_4$ improved to 81.78%, an $\sim$ 4% improvement. Compared to MC-Spatial Dropout and SpinBayes methods, the OOD detection rates are generally similar. In the worst case, the OOD detection rate is $\sim$ 14% lower for the VGG topology on the $\hat{\mathcal{D}}_6$ dataset. Therefore, the results indicate that the proposed MC-Scale Dropout method is a robust and reliable solution to OOD detection across diverse model architectures and datasets.

Epistemic Uncertainty of Semantic Segmentaion For biomedical segmentation tasks, the epistemic uncertainty is calculated for each pixel. The fifth and eighth columns of Fig. 4.25 depict the pixel-wise uncertainty masks (qualitatively) for the MC-Dropout and the proposed MC-Scale Dropout method. In segmentation tasks, an ideal model would produce high uncertainty around misclassified pixels and low uncertainty around correctly classified pixels. Overall, as depicted in Fig. 4.25, the uncertainty is high around the misclassified pixels for the proposed method, but correctly classified pixels have low uncertainty. In general, the uncertainty masks for MC-Dropout are darker, depicting slightly stronger uncertainty estimates due to their higher model precision (32 bits) and a higher Dropout probability (50%). However, in some cases, the uncertainty mask is also stronger in the region of correctly classified pixels. However, our proposed method produces uncertainty only around miss-classified pixels.

Table 4.20.: Energy Efficiency Comparison of Hardware Implementations

Related works	Technology	Topology	Bit resolution	Energy
H. Fan et al.[96]	FPGA	ResNet18	8-bit	0.014 J/Image
R.Cai et al.[168]	FPGA	3-FC	8-bit	18.97 µJ/Image
X.Jia et al.[169]	FPGA	3-FC	8-bit	46.00 µJ/Image
H.Awano et al. [95]	FPGA	3-FC	7-bit	21.09 µJ/Image
A. Malhotra et al. [99]	RRAM	3-FC	4-bit	9.30 µJ/Image
S.T.Ahmed et al.[84]	STT-MRAM	LeNet-5	1-bit	2.00 µJ/Image
S.T.Ahmed et al.[86]	STT-MRAM	LeNet-5	1-bit	0.68 µJ/Image
K.Yang et al.[102]	Domain wall-MTJ	3-FC	4-bit	0.79 µJ/Image
Proposed implementation (MNIST)	**SOT-MRAM**	**LeNet-5**	**1-bit**	**0.18** µJ/Image
Proposed implementation (CIFAR-10)	**SOT-MRAM**	**VGG**	**1-bit**	**0.29** µJ/Image

4.3.4.5. Hardware Overhead Analysis

To assess the energy consumption of the proposed approach, we estimated the required resources for implementing a network of five layers with the Scale-Dropout method, and we assumed using 10 crossbar arrays of 256×256 and 10 Spin-ScaleDrop modules to implement a LeNet-5 network. The total area needed for the implementation of the LeNet-5 topology is $0.401\,mm^2$, comprising the crossbar arrays and the memories. The area estimation is based on the NVSim and layout measurement. Given the energy consumption of the different components of our architecture shown in Table 4.14. We used the NVSim simulator to estimate the total energy consumption for an inference run and multiplied this value by the number of forward passes (MC run). The analysis is carried out for ten forward passes ($T = 10$). The energy consumption of an inference run is shown in Table 4.20 compared to other FPGA and CiM implementations. We evaluated two topologies, LeNet-5 for the MNIST dataset and VGG-9 (9 layers) for CIFAR-10. For a consistent benchmark, the same metrics as in previous studies were used. The Scale-Dropout approach significantly improves energy efficiency, reaching up to 100× higher efficiency compared to the method presented in [95]. Compared to the implementation in [99], our approach is 51× better. Furthermore, compared to the implementation based on STT-MRAM [86], the proposed approach exhibits 3.77× better efficiency. Finally, compared to reference [102], our approach demonstrates 4.38× greater energy efficiency. To scale up the approach, we have performed an energy consumption estimation with a VGG-9 (9-layers) topology, and we report 0.29 $\mu J/Image$. Thus, energy consumption remains notably low even when considering a larger dataset such as CIFAR-10 and the VGG topology.

Furthermore, Scale-Dropout requires only one RNG per layer compared to similar approaches [84, 85, 86], as shown in Table 4.19. An RNG can be shared for all layers to reduce the number of RNGs for the entire model to one. Consequently, a reduction in dropout modules by $K * K * C_{in}\times$ compared to SpinDrop [85] and $C_{in}\times$ compared to Spatial-SpinDrop [86] work, assuming $C_{in} = C_{out}$. This significantly contributes to a reduction in energy consumption and chip area. Specifically, the chip area for a layer is reduced by up to 229×, and power consumption is reduced by up to 2304×.

4.3.4.6. Discussion

In Distribution Uncertainty Analysis We thoroughly analyzed the performance of the proposed method in data distribution shift and out-of-distribution data in Section 4.3.4.4. However, it is equally important to perform well when it receives in-distribution data. This means that correct predictions should have low uncertainty, and a model should accept most of them.

In our in-distribution data analysis (Table 4.21), we present the accepted, rejected, TPR, TNR, and AR percentages. TPR indicates the rate of correct and accepted predictions, while TNR refers to rejected and incorrect predictions. High TPR and TNR rates are desired as they suggest that most of the accepted

Table 4.21.: Analysis of the proposed method using in-distribution data, showing True Positive Rate (TPR), True Negative Rate (TNR), and Acceptance Rate (AR) for various topologies.

Topology	Accept	Reject	TPR	TNR	AR
VGG	77.43%	22.57%	84.29%	81.67%	97.45%
ResNet-18	77.48%	22.52%	84.00%	81.83%	97.67%
ResNet-20	41.00%	58.98%	59.24%	95.83%	99.12%

predictions have low uncertainty, and incorrect predictions have high uncertainty. A high AR rate also indicates that most of the correct predictions are accepted.

The VGG and ResNet-18 models, with their larger size, effectively handle the complexity of the CIFAR-10 task, showing acceptance of approximately 80% and more than 80% in both TPR and TNR, plus more than 97% in AR, confirming the efficacy of our method.

On the contrary, the smaller ResNet-20 model is not optimal for handling the complexity of CIFAR-10, leading to 'uncertainty in model architecture' [188] and consequently to greater uncertainty in prediction. To be specific, its inference accuracy is comparatively lower $\sim$ 86% compared to $\sim$ 91% for the other model, since it only has 16, 32, and 64 neurons in the residual blocks. Thus, it has a lower acceptance rate (41%). That means that most predictions are uncertain, and our method is also effective in quantifying 'uncertainty in the model architecture.'

Note that our classification of the predictions (OOD or ID) with our approach is conservative and prioritizes certainty, i.e., only certain predictions reach the end user. Adjusting quantile and confidence scores can increase acceptance rates closer to inference accuracy but may decrease OOD detection rates.

Corruption Robustness Analysis The proposed method is evaluated on 15 common corruptions reported in the work (CIFAR-10-C) [20] with various topologies with and without pre-processing, as shown in Table 4.22. Our approach can achieve an OOD detection rate of on average 87.06%, 86.10% and 97.64% for VGG, ResNet-18, and ResNet-20 topologies, respectively, when no pre-processing is applied.

On the other hand, when the corruption robustness dataset is pre-processed by channel-wise normalizing them, i.e., they have the same channel-wise distribution as the clean CIFAR-10 data the model expects, the corruption error drastically reduces. For example, the mean corruption error for VGG was reduced from 82.84% to 49.95%. Consequently, the uncertainty of the predictions also reduces. Specifically, our approach achieves OOD detection rates of 58.48%, 56.21%, and 87.73%, respectively, for VGG, ResNet-18, and ResNet-20 topologies. Therefore, pre-processing the dataset standardizes the data and improves the corruption robustness similar to the histogram equalization method as discussed in [20].

In terms of topology, in larger networks, e.g., ResNet-18, the corruption error is relatively lower. For example, in the case of Gaussian noise, the corruption error is reduced from 86.85% in VGG to 83.52% in ResNet-18. A similar trend is observed for other datasets. However, despite the fact that the ResNet-20 model is smaller than ResNet-18, it has a relatively higher corruption error because the smaller model introduces "uncertainty in model architecture" as mentioned in the previous section.

Nevertheless, there is a direct relationship between corruption error, uncertainty, and, in turn, the OOD detection rate. In cases where the accuracy is reduced by a small margin, the model uncertainty is low, and the OOD detection rate with our approach is also low. For example, the worst-case OOD detection rate for the VGG topology is 29.53%. This is achieved when the accuracy is reduced by only 6.89% for brightness corruption. On the other hand, the highest OOD detection rate is achieved when the accuracy is reduced by 78.88% for VGG topologies.

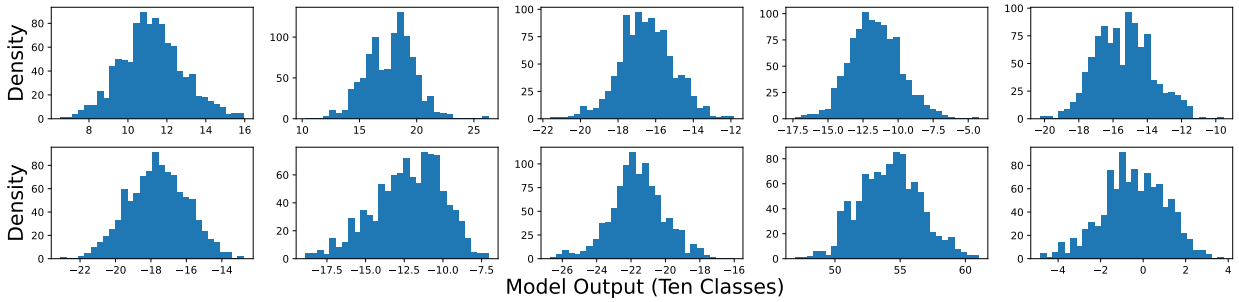


Figure 4.26.: Per class posterior distribution of ResNet-18 topology with a Monte Carlo sample size of 1000.

Table 4.22.: Analysis of mean corruption errors (mCE) and mean out-of-distribution detection (mOOD) detection values of different topologies when various corruptions applied CIFAR-10 with and without pre-processing (PP). All numbers represent percentages.

					Noise			Blur				Weather				Digital			
Topology	PP	Error	mCE	mOOD	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
VGG	No	9.55	82.84	87.06	80.57 (86.85)	80.08 (86.27)	71.6 (61.07)	91.41 (86.91)	89.99 (86.41)	88.95 (86.03)	95.24 (86.45)	76.38 (82.33)	87.52 (85.51)	97.42 (88.43)	86.23 (70.71)	84.98 (90.16)	95.26 (82.59)	87.47 (81.12)	92.85 (81.83)
	Yes		49.95	58.48	68.39 (70.45)	67.96 (66.54)	78.78 (87.85)	62.75 (50.72)	59.96 (50.8)	55.78 (43.25)	58.97 (46.06)	47.0 (34.86)	56.7 (46.96)	58.09 (41.78)	29.53 (16.44)	89.84 (78.31)	48.13 (30.07)	47.38 (55.28)	47.95 (29.93)
ResNet -18	No	8.48	79.49	86.10	84.38 (83.52)	83.57 (82.03)	82.66 (84.26)	95.89 (82.13)	89.60 (83.06)	93.37 (79.46)	94.48 (81.61)	80.01 (72.81)	89.97 (80.81)	98.64 (82.34)	73.90 (61.02)	82.66 (84.26)	86.98 (78.20)	74.18 (81.28)	81.25 (75.56)
	Yes		47.32	56.21	66.53 (66.71)	60.42 (65.96)	73.73 (59.13)	62.75 (50.72)	58.90 (49.82)	59.54 (44.34)	61.11 (47.05)	47.7 (32.58)	53.78 (59.25)	58.09 (41.78)	28.79 (14.61)	65.71 (74.63)	47.98 (26.66)	55.15 (53.48)	42.98 (23.02)
ResNet -20	No	13.96	74.38	97.64	99.57 (85.75)	99.44 (84.97)	84.38 (16.48)	100.0 (83.23)	99.96 (85.86)	100.0 (82.70)	100.0 (83.13)	82.66 (15.74)	99.98 (84.45)	100.0 (87.63)	99.73 (73.72)	100.0 (85.62)	100.0 (83.22)	99.05 (82.67)	99.88 (80.53)
	Yes		55.61	87.73	77.74 (84.34)	81.80 (82.06)	88.09 (76.32)	96.03 (57.51)	88.46 (61.83)	94.31 (50.03)	94.50 (50.95)	85.51 (41.56)	89.11 (53.56)	93.78 (47.58)	69.94 (20.31)	99.95 (78.39)	90.30 (36.82)	83.92 (60.79)	82.46 (32.16)

Variability and scalability This study builds on previous research demonstrating the robustness of the dropout approach against device variability [85]. It highlights the impact of dropout module variations on network accuracy as shown in Table 4.16. Moreover, we propose to utilize SOT-MRAM, capable of achieving resistance levels up to several MΩ, which aligns with previous simulations emphasizing resistance's crucial role in constructing large arrays [189], validating the scalability and energy efficiency of this approach. In the work [190], different non-idealities of Spintronics devices were evaluated to assess their impact on uncertainty estimates and accuracy. Our proposed Scale-Dropout approach shows good resilience against different faults compared to other binary BayNN [85].

Empirical Analysis of the Posterior Distribution We have performed an empirical evaluation of our method on the CIFAR-10 dataset and the ResNet-18 topology. We have observed that as the number of Monte Carlo samples increases, the histogram of the posterior distribution for each of the 10 classes indeed approaches a Gaussian distribution and can be considered as an approximate Gaussian distribution similar to the MC-Dropout method [35].

4.3.5. Scientific Impact of This Work

We outline the scientific impact of this work and our main contributions:

1. **Vector Dropout and Resource Scalability:** This work aims to reduce the number of Dropout modules in hardware to only one regardless of the model size, the lowest one can get for Dropout-based BayNNs. Consequently, the energy consumption and chip area for the Dropout modules can be reduced by 2304× and 229×, respectively, compared to our previous works [84, 86, 85]. Therefore, to improve resource scalability in edge devices, research in the direction of vector-wise Dropout is a viable candidate.

2. **Signal Flow Consideration:** The signal flow through the NN should be carefully considered when it comes to vector Dropout. A naive vector dropout will lead to total information loss for the following layers. In this work, a novel approach, called "Unitary Dropout", is proposed which effectively drops the desired vector, scale vector, to ones instead of zero. Consequently, the scale vector is ignored when dropped, but still allows the signal to flow through the NN.

3. **Alternate Dropout for Bayesian Inference:** The work proposed Scale-Dropout-based Bayesian Inference for efficient uncertainty estimation in BNNs. The evaluation shows highly effective uncertainty estimates when in uncertain conditions. Therefore, alternative Dropout methods are also a viable solution for uncertainty estimation.
4. **Adaptive Dropout Rates:** Instead of applying the same dropout rates to all layers, a layer-wise adaptive dropout rate allows a Dropout to be applied to all layers in the NN. As a general rule, a low dropout rate should be applied to layers with low parameter counts. For larger layers, the dropout rate should be relatively higher.
5. **Real-World Applications:** The work further improved in energy efficiency and performance metrics of our previous works [85, 84, 86] and related works. Therefore, the applicability of BayNNs to edge devices with limited resources or to real-time applications is further reduced. Nevertheless, it allows effective decision-making under uncertainty.

4.3.6. Section Conclusion

In this section, we propose a novel Dropout approach, Scale-Dropout, which drops the entire scale vector of a layer stochastically with a probability p. Our approach required only one Dropout module in the hardware, regardless of the model size. Additionally, we propose scale-Dropout-based Bayesian inference, MC-Scale Dropout, for efficient uncertainty estimation. Furthermore, we propose a novel SOT-MRAM-based CiM implementation for accelerating Bayesian NNs. In our CiM architecture, the stochastic and deterministic aspects of SOT-MRAM have been combined in a crossbar array-based architecture, with only changes made in peripheral circuitry, and achieve up to 100× of energy savings. In terms of uncertainty estimation, our approach can detect up to 100% out-of-distribution data, significantly higher uncertainty estimates compared to popular uncertainty estimation approaches. Additionally, predictive performance is improved by up to 1.33% compared to SOTA binary BayNN and improved by up to 0.26% compared to conventional BNN approaches. Our approach combines the algorithmic approach with the cost-effective and energy-efficient SOT-MRAM-based CiM implementation for reliable prediction.

5. Variational Inference-Based Bayesian NNs

In terms of inference accuracy and quality of uncertainty, Variational Inference (VI) typically offers a more precise inference and a more accurate representation of uncertainty. However, the challenges lie in their implementation in the CiM architecture due to their distributional parameters and on-the-fly sampling requirements. Another drawback of conventional VI applications is their memory consumption. In this chapter, we present several approaches proposed in this thesis with full-stack optimization for VI-based BayNN.

5.1. Bayesian In-Memory Approximation and CiM-Aware NN Architecture for Efficient Sampling

In this section, we first address the limitations of BNNs in terms of computing cost and CiM implementation and discuss our proposed method at the algorithm level. Finally, the approach for hardware implementation and CiM-based Bayesian inference are described. We propose a novel BayNN topology, memory-centric approximation of BayNN, BayNN inference method, mapping strategy, and CiM architecture.
This section is based on our journal publication in ACM TECS, 2023 [104].

5.1.1. Methodology

5.1.1.1. Bayesian In-Memory Approximation

A memory-centric approximation is desired to enable efficient mapping of the posterior distribution in the CiM architecture. Therefore, we propose an approach called *Bayesian in-memory approximation*. Our proposed *Bayesian in-memory approximation* method provides a memory-friendly posterior distribution $\hat{q}_{\omega}(\theta)$ that can be efficiently mapped and sampled by the CiM hardware that implements BayNN. We empirically demonstrate later that the proposed approximate posterior distribution $\hat{q}_{\omega}(\theta)$ is similar to the original variational distribution $q_{\omega}(\theta)$ in terms of predictive performance and uncertainty estimation.

In the proposed *in-memory approximation*, we first discretize the variational distribution $q_{\omega}(\theta)$ into J possible samples. We refer to this discretized version as $\hat{q}_{\omega}(\theta)$. Each element of $\hat{q}_{\omega}(\theta)$ represents a full-precision value from $q_{\omega}(\theta)$ without replacement. This discretization is performed before deploying the proposed BNN to CiM. It is important to note that this approximation only allows the representation of $q_{\omega}(\theta)$ with independent parameters. The consequence of this approximation is that, during Bayesian inference, more frequent sampling from $q_{\omega}(\theta)$ is possible. However, this is a common modeling choice in practice.

Afterward, each value of the discrete distribution $\hat{q}_{\omega}(\theta)$ is quantized to fit the limited stable states of the spintronic device used for storage. This is a common practice in hardware implementations of NN; otherwise, inference accuracy could degrade due to quantization error [191]. To achieve this, we propose a post-training and pre-mapping quantization method that quantizes $\hat{q}_{\omega}(\theta)$ into υ-bit values, as follows:

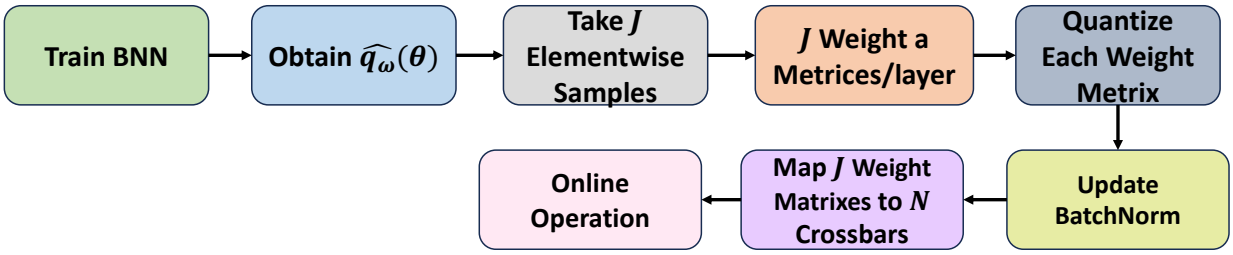


Figure 5.1.: Overall flow of the proposed *Bayesian in-memory approximation* for efficient mapping of Bayesian distributions to CiM architectures.

$$\hat{q}_\omega(\theta) = \text{round}\left(\text{clip}(\hat{q}_\omega(\theta), -1, 1) \times \left(2^{v-1} - 1\right)\right), \tag{5.1}$$

where clip(., min, max) clamps all elements of the input it receives within the range [min, max], and round(.) rounds each element to the closest integer. Alternatively, other quantization functions, such as DoReFa [192], can also be used. However, it is important to note that during training, a small prior for $\boldsymbol{\sigma}_\omega^2$ must be chosen, since we use clip(., min, max) in the post-training quantization process. Otherwise, this clamping function should be applied during training as a regularizer.

Our quantization approach quantized all the sampled parameters into fixed-point values. Otherwise, mapping quantized values to spintronic conductance states would require a look-up table (LUT) for each parameter. This is because the distribution of the parameters differs from each other, meaning that the mean and variance of weights vary between the parameters, resulting in different value ranges for different parameters. However, with our quantization approach, only one LUT is required.

Usually, normalization layers standardize the neuron distribution during training. However, the proposed quantization can significantly alter the neuron output distributions relative to the trained distributions. In our evaluation, we have found that this mismatch can result in drastic accuracy degradation. Thus, we propose a re-calibration method to bring the post-quantized neuron distributions closer to the training distributions. Specifically, we adjust the variables, the mean and variance, in the normalization layers, such as batch normalization. This adjustment is made by estimating the variables based on several mini-batches of training data using the quantized parameters $\hat{\boldsymbol{\theta}}$. The resulting distribution $\tilde{q}(\boldsymbol{\theta})$ is considered an approximation of the variational distributions $q(\boldsymbol{\theta})$.

Lastly, each of the quantized samples is mapped to J parallel crossbars for a layer, resulting in each layer being represented by J parallel crossbars. Since the overall approximation aims to implement the Bayesian neural network in in-memory architectures, we refer to it as the *Bayesian in-memory approximation*. The overall flow of the proposed *Bayesian in-memory approximation* is depicted in Fig. 5.1.

5.1.1.2. Design Space Exploration Optimizing Bit-precision

There is a trade-off between the quantization level, which influences the design of the bit-cell, and the resulting performance. Using a large quantization level, e.g., 8-bit, can lead to better performance because it has better representation capabilities. However, it leads to a large number of memory cells, power consumption for inference (as more memory cells require reading), and chip area. Our collaborator has designed multi-bit Spintronic-based memory cells to represent $\tilde{q}(\boldsymbol{\theta})$. The design of memory cells also becomes challenging with larger quantization level.

Therefore, to determine the optimal quantization levels, a design-to-technology space exploration was performed. Based on the analysis, the multi-bit memory cells were designed.

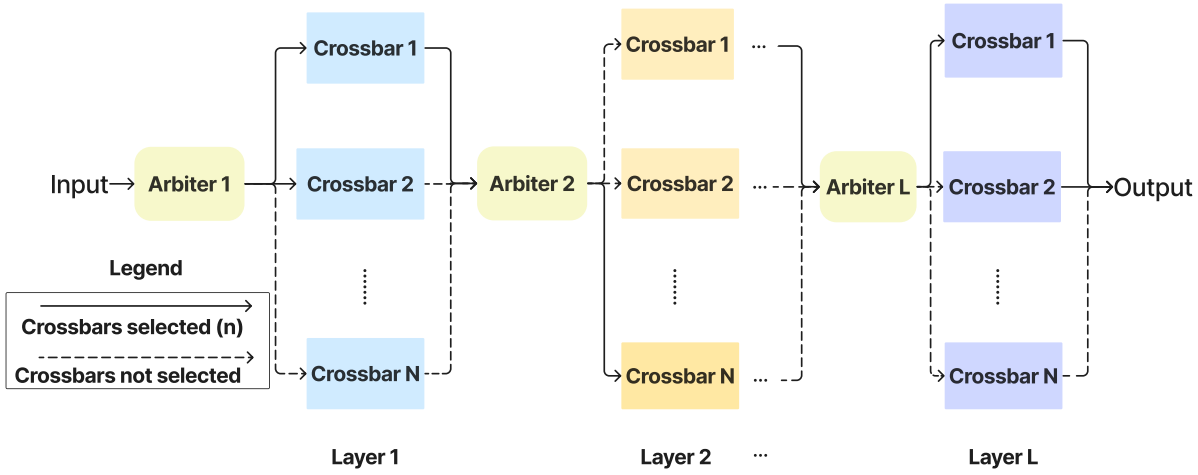


Figure 5.2.: General SpinBayes topology. An arbiter is utilized in each layer to determine in which crossbar the MAC operation is performed.

5.1.1.3. Efficient Sampling from CiM Architectures

Although our proposed Bayesian in-memory approximation allows mapping the posterior distribution to the CiM hardware, sampling from the distribution remains a challenge. To address this, we propose a novel network topology called *SpinBayes*. The general topology of SpinBayes is depicted in Fig. 5.2. In the SpinBayes topology, an Arbiter is utilized for each layer to select n (where $n = 1$ to $J - 1$) out of the total J crossbars for n-way Bayesian inference. During each forward pass of Bayesian inference, the Arbiter generates a random binary one-hot vector of length J, where "0" represents an unselected crossbar and "1" represents the selected crossbar. Each selected crossbar for layer l receives the same input z_{l-1}, which is the activation of the previous layer, for the MAC operation. If $n \geq 2$, the results of the MAC operations for the selected crossbars are averaged. For example, in the case of the $n = 2$ way Bayesian inference where crossbars at index 1 and 3 are selected by the Arbiter at random, the MAC operations for a layer can be depicted as:

$$z_l = \frac{1}{n}(z_{l-1}\mathbf{W}_{1,l}^{\top} + z_{l-1}\mathbf{W}_{3,l}^{\top}). \tag{5.2}$$

Where $w_{1,l}^{T}$ and $w_{3,l}^{T}$ represent the weight matrices at index 1 and 3, respectively, z_l is the resulting intermediate activation of layer l, and $(.)^{\top}$ represent matrix transpose operation. Afterward, z_l is processed by subsequent layers such as pooling, Batch normalization, and non-linear activation to produce the final activation of layer l. The overall computational flow for a layer is depicted in Fig. 5.3. However, for a 1-way Bayesian inference ($n = 1$), averaging is not required, as the MAC operation is performed on only one crossbar array. In this case, assuming the crossbar at index 1 is selected by the Arbiter at random, the MAC operation can be depicted as:

$$z_l = z_{l-1}\mathbf{W}_{1,l}^{\top}. \tag{5.3}$$

Ultimately, the Bayesian inference with our proposed SpinBayes topology for obtaining final network predictions $\mathbf{y}^*$ given the input $\mathbf{x}^*$ is expressed as:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p\left(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}^{(t)}\right)$$

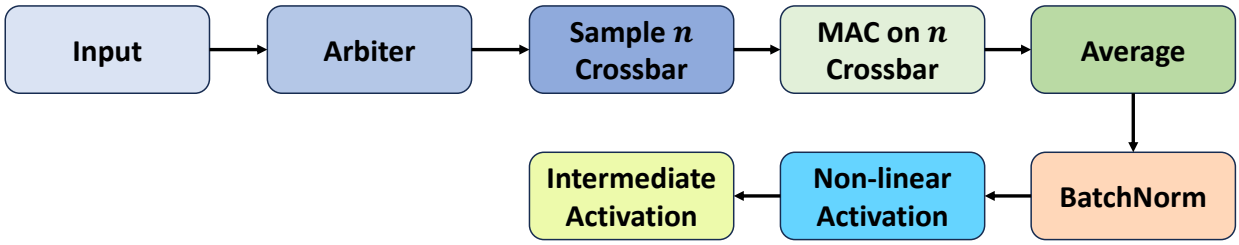


Figure 5.3.: The flow depicting computation carried out in a layer of the proposed *SpinBayes* implemented on CiM architectures.

$$\text{with} \quad \boldsymbol{\theta}^{(t)} \sim \tilde{q}(\boldsymbol{\theta}) = \text{Choose}(\mathcal{S}). \tag{5.4}$$

Here, the given $\mathbf{x}^*$ is passed through the SpinBayes network T times, and the final network predictions are averaged to give the prediction $\mathbf{y}^*$. In each forward pass, the Arbiter randomly samples n crossbar arrays (weight matrices) layer-wise for the MAC operation (as described earlier) to produce a stochastic output each time.

In addition, the resulting distribution for the parameters of the proposed BNN can be represented as $\tilde{q}(\boldsymbol{\theta}) := \text{Choose}(\mathcal{S})$, where the set $\mathcal{S}$ denotes the set of all parameter combinations that can be drawn this way, and $\text{Choose}(\cdot)$ denotes a uniform selection from $\mathcal{S}$. Specifically, $\mathcal{S} = \{\boldsymbol{\theta}^{(s)} \sim q_{\omega}(\boldsymbol{\theta}) \mid s = 1, \ldots, S\}$.

Furthermore, the proposed SpinBayes topology can be generalized to all existing CNN topologies by making minor modifications to the network topologies. For popular topologies such as ResNet and VGG, an Arbiter and an averaging block can be inserted before and after the convolutional layer, as shown in Fig. 5.4. The MAC is performed concurrently on all selected convolutional layers.

Note that n-way inference with $n \geq 2$ may lead to an underestimation of uncertainty due to averaging, but it has been empirically shown to improve inference accuracy, as demonstrated in Section 5.1.3. In the case of n-way inference, the effective size of $\mathcal{S}$ depends on the number of crossbars n chosen in the forward pass. Generally, for n-way inference and L layers of crossbars, there are $\binom{J}{n}^L$ possible combinations in $\mathcal{S}$. In Section 5.1.5.5, we discuss the overhead associated with $n \geq 2$-way inference.

5.1.1.4. Bayesian Inference on CiM Architecture

To implement the proposed Bayesian posterior distribution, we propose a novel spintronic-based CiM architecture. The architectural design is shown in Fig. 5.5. The spintronic architecture is designed around a multi-value bit-cell crossbar array. The quantized weights are stored in the multi-value cell made with four MTJs, as shown in Fig. 5.8(a). The number of conductance levels depends on the number of MTJs, and in our design, we limit the number of MTJs to four for reliable reading and writing. Each multi-level device has five reliable levels of conductance: 4AP, 3AP-1P, 2AP-2P, 1AP-3P, and 4P. To achieve more conductance levels, it is possible to use multiple multi-level cells jointly for the storage of quantized weights (v-bit) [195].

The next section describes the circuit-level behavior of the multi-value implementation. In our architecture, the arbiter, as described earlier, is implemented in the periphery of crossbars within a layer, and utilizes the stochastic behavior of the spintronic devices. The arbiter randomly selects n crossbars per layer that receive the same input. The choice of n determines the architectural implementation. In our case, we have chosen $n = 1$. Each crossbar has two decoders, one for the reading operation and the other for the writing operation, which enable the activation of multiple addresses. The spintronic arbiter is connected to the enable signal of the reading decoder of the different crossbar arrays.

The MAC operation in each crossbar is performed by activating multiple wordlines in the array. This allows the current in the bit-line of each crossbar to be sensed by a flash analog-to-digital converter. To sum up and accumulate the contributions from different crossbars, a CMOS-based Adder-Accumulator (AAC)

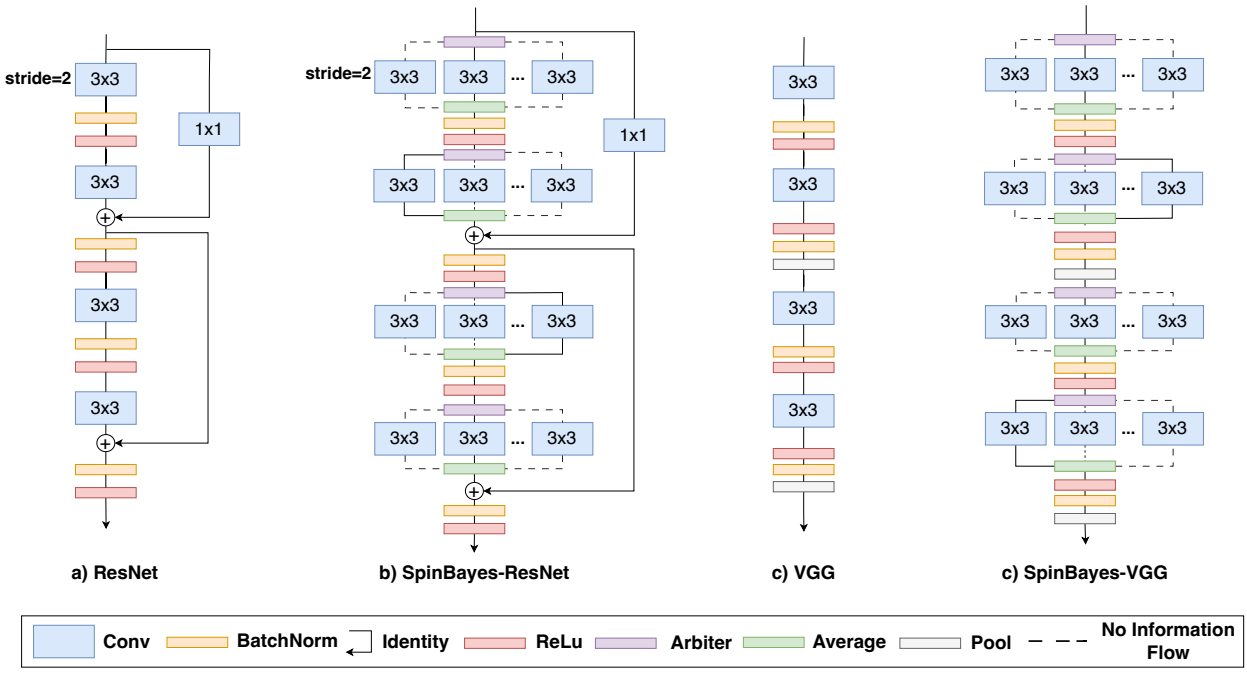


Figure 5.4.: Sketch of proposed SpinBayes based on popular CNN topologies ResNet [193] and VGG [194]. Here, we only show the first four layers of a specific topology. Our proposed SpinBayes topology is generalizable across most existing topologies, with only the addition of arbiter and average blocks required. MAC operation on n-way inference with $n > 1$ parallelized in each layer. **It is recommended to view this figure in color.**

is utilized. At the output of the crossbar, a CMOS circuit computes layer-wise average weighted sums as described in Section 5.1.1. The activation functions, although not shown in Fig. 5.5, are implemented with comparators to obtain the overall activation of the layer before passing it to the subsequent layers. Similarly, the same averaging circuits are used at the final layer of the SpinBayes network to estimate the prediction based on the T stochastic forward passes, as expressed in equation equation 5.4.

Each of the J crossbars contains a unique configuration of weights. Consequently, the resulting current at the output of the crossbar varies depending on the specific crossbar selected. Therefore, the required mean and variance are translated through the distribution of current at the output of the crossbar.

5.1.2. Hardware Implementation

5.1.2.1. Spintronic Arbiter

One spintronic-based Arbiter is implemented for each neural network layer to enable $n = 1$ selection in the n-way crossbar selection for each layer in the SpinBayes topology. In this implementation, the stochastic regime of an MTJ is utilized as a random number generator. The probability density function for switching the SOT-MTJ follows an exponential distribution and is expressed as [180]:

$$p_{sw} = 1 - \exp\left(\frac{t}{\tau}\right) \tag{5.5a}$$

$$\tau = \tau_0 \exp\left[\frac{\Delta}{k_B \mathcal{T}}\left(1 - 2\frac{I}{I_{c0}}(\frac{\pi}{2} - \frac{I}{I_{c0}})\right)\right] \tag{5.5b}$$

Here, Δ is the thermal stability factor, I is the applied current through the SOT-track, t is the pulse duration, τ_0 is the attempt time, I_{c0} is the critical current at 0 K, k_B is the Boltzmann constant, and $\mathcal{T}$ is the temperature. I_{c0} represents the minimum current required to switch the MTJ. The equation equation 5.5 is

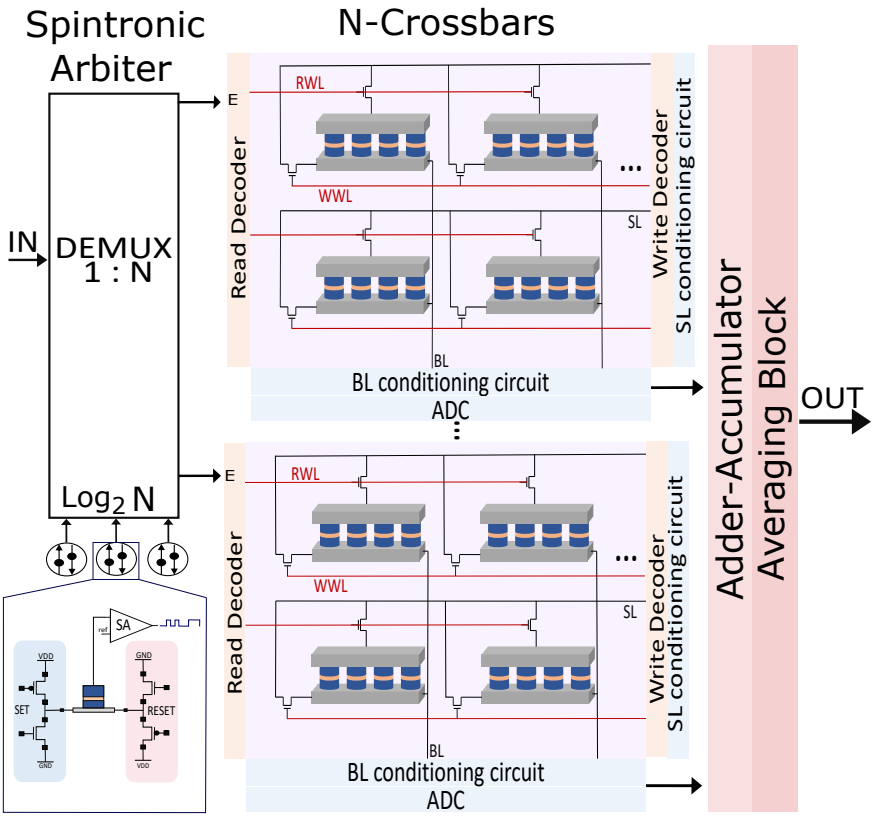


Figure 5.5.: Proposed layer architecture

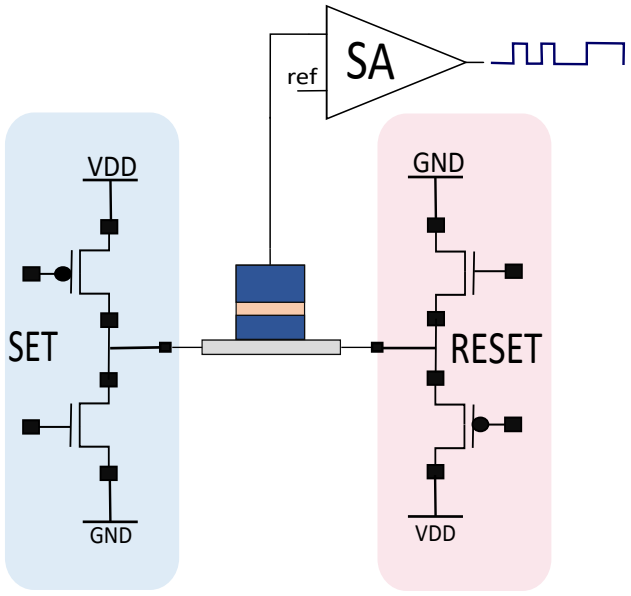


Figure 5.6.: Spin-Orbit Torque random number generator

used to model and evaluate the switching behavior of the SOT-MTJ for different switching currents, while keeping the pulse width fixed at 10 ns. The results of this evaluation are shown in Fig. 5.7. It is important to note that in order to achieve a switching probability of 50% for the MTJ, a current of 230 µA is required. To generate the bidirectional current across the SOT track, four transistors are added, as shown in Fig. 5.6. The desired switching probability of 50% is achieved by programming the MTJs through successive "SET" and "RESET" operations using a current of 230 µA.

To ensure reliable switching of the MTJ, the write duration is set to 10 ns for the SET operation and 5 ns for the RESET operation. The state of the MTJ is read using a sense amplifier. The SET and RESET cycles are

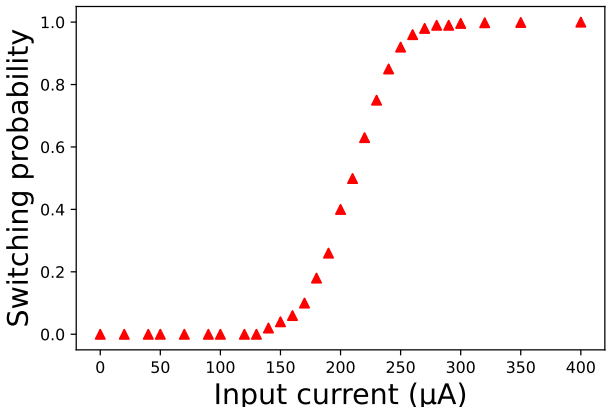
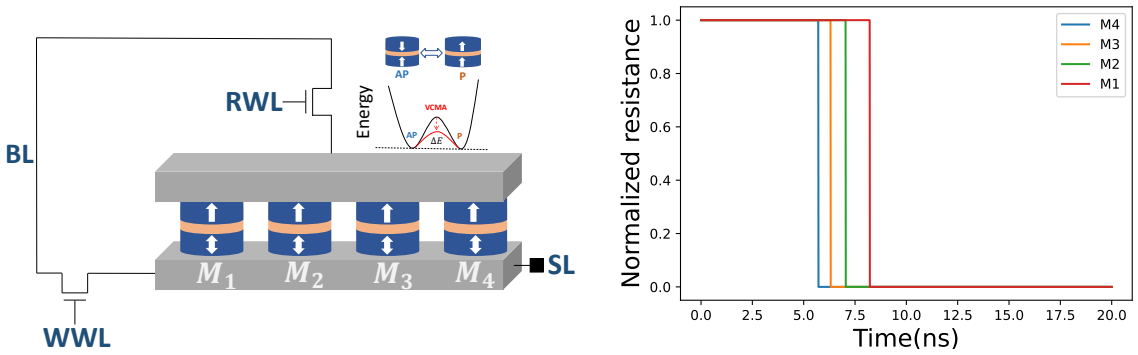


Figure 5.7.: The probability of switching the stochastic device of the spintronic Arbiter



(a) Multi-value cell with 4 MTJs

(b) Multi-value cell switching

Figure 5.8.: Proposed multi-value SOT bit-cell. **It is recommended to view this figure in color.**

repeated to generate a stochastic sequence, which is then passed through a DEMUX (1-to-J) circuit to enable the selection of the read word-lines in the crossbar array. For the selection of crossbars, $\log_2 J$ stochastic spintronic selection circuits are utilized to control the DEMUX. The stochastic behavior of the MTJ serves as a random switch selector for the DEMUX, ensuring the random selection of the J crossbars.

5.1.2.2. SOT multi-value cell design for Quantized Weights

The design of the multi-value cell for representing the quantized parameter values is inspired by [196]. Multiple MTJs are implemented in parallel to emulate a multi-level spintronic memristor utilizing the VCMA and SOT effects (see Section 2.8.1). The overall cell design is depicted in Fig. 5.8 (a). The VCMA effect, which depends on the voltage across the MTJ, reduces the switching barrier and facilitates the writing for a given SOT current. In conventional MRAM devices, the energy barrier is typically set high to prevent thermal noise and accidental writing during reading. However, a high energy barrier requires a higher current to switch the MTJ. The VCMA effect can be utilized to lower the barrier and reduce the writing current [197]. It is worth noting that the multi-value cell design requires only two access transistors for read and write operations, compared to the traditional MTJ implementation where each MTJ would require two transistors. This leads to more efficient utilization of hardware resources and a denser cell structure. The access transistors are controlled by a Write Word-Line (WWL) and a Read Word-Line (RWL). By adjusting the RWL voltage, individual MTJs or multiple MTJs can be programmed simultaneously, increasing the versatility of the cell utilization.

In this structure, each MTJ experiences a different voltage drop depending on its position on the SOT track. For example, $\mathbf{M}_4$ experiences a higher voltage drop than $\mathbf{M}_1$, resulting in a reduced switching time for this specific MTJ, as shown in Fig. 5.8 (b). To increase the number of levels considered for computation, a

differential conductance pair scheme is employed, enabling the representation of 9 levels as demonstrated in [196] for ternary implementation. However, for the required $v = 4$-bit quantization range, multiple multi-value cells need to be used in conjunction [195].

In the proposed crossbar design, *both the stochastic and deterministic behaviors of the SOT-MRAM are exploited*. Fig. 5.7 illustrates the relationship between the probability of device switching and the applied current. To utilize the device as a stochastic element, a programming current of approximately 230 µA is required, resulting in a 50% probability of switching. However, for weight storage purposes, the writing current must exceed 300 µA. By increasing the current beyond 300 µA, the probability of switching the device approaches 1, minimizing the likelihood of write failure in the CiM architecture.

In the design simulation, a writing current of 500 µA is used to program the MTJ. The Read Word-Line (RWL) voltage is varied from 0 to 1 V with a period of 10 ns. Fig. 5.8(b) illustrates the switching times of the different MTJs (represented as $\mathbf{M}_x$) for an RWL voltage of 1 V. It is noteworthy that all the MTJs exhibit switching behavior at this specific voltage.

The resistance values (R_P and R_{AP}) are calibrated within the range of 2.5 MΩ to 5.5 MΩ to achieve low output current. To ensure reliable sensing of the cells at the output of the crossbar, we activate four cells at a time and perform the current sum. This allows for parallel sensing of a limited number of cells in the crossbar arrays.

5.1.3. Evaluation

Here, we thoroughly evaluate our proposed approach from both an algorithmic and hardware implementation perspective. However, it is important to note that BNNs are fundamentally different from conventional NNs and that they inherently require more resources to perform the Bayesian inference with the additional benefit of uncertainty information. Therefore, the proposed method is only compared with the related BNN methods.

Additionally, the proposed approach deals with the implementation of emerging technology for the implementation of BNNs. Thus, no real hardware is available and software-based simulations are employed. For algorithmic results, training and evaluations are conducted using the PyTorch framework. For the hardware-related results, only electrical simulations are performed for this approach. Specifically, the crossbar simulations were carried out on Spice, incorporating the concepts presented earlier in Section 5.1.2.1 and Section 5.1.2.2. Both models, the stochastic and the multi-value cell, have similar physical parameters, and *we exploit the different mechanisms of spintronic devices for our implementation.* However, parameters such as values and shapes of weights were transferred from the algorithmic level. Weight transfer on the crossbar enables us to evaluate accuracy at the hardware level and assess any potential drop in accuracy that may occur.

5.1.4. Simulation Setup

The proposed method is evaluated on both classification tasks with up to 100 classes and semantic segmentation tasks, including binary and multi-class segmentation. Semantic segmentation, which involves segmenting an image into multiple sections and assigning each pixel with its corresponding class label, is known to be more challenging than classification tasks due to its finer granularity.

For the classification tasks, we evaluate the proposed method using LeNet, VGG (based on [36]), MobileNet, and ResNet topologies. These topologies have up to 34 layers and 21.79×10^6 parameters. The models are trained on MNIST, CIFAR-10, and CIFAR-100 datasets as the in-distribution datasets ($\mathcal{D}_{in}$).

MNIST is a well-known dataset for handwritten digit recognition, while CIFAR-10 and CIFAR-100 present more challenging classification tasks. CIFAR-10 contains images from 10 classes, while CIFAR-100 expands this to 100 classes, significantly increasing the difficulty of the classification assignments. Despite the

relative ease of classifying the MNIST dataset, evaluating our method on it is necessary for appropriate comparison with related works.

For the semantic segmentation tasks, we evaluated the proposed method on two safety-critical tasks: real-world medical image diagnosis using Kvasir-SEG [198] and skin lesion segmentation [182], as well as automotive scene understanding using the CamVid [199] dataset. Kvasir-SEG contains medically obtained gastrointestinal polyps, while the skin lesion dataset contains microscopic skin lesions. Both medical datasets have two classes, and the task is to segment the regions of interest within the images. The CamVid dataset consists of road scene images and involves segmenting each pixel into one of 12 classes, making it a significantly more challenging task. For all semantic segmentation tasks, we used the Feature Pyramid Network (FPN) architecture with ResNet-18 as the encoder. The BNN parameters were quantized to 4 bits (Q=4) across the entire topology for all tasks.

To evaluate the uncertainty of the model, we use the correct-certain ratio (R_{cc}), incorrect-uncertain ratio (R_{iu}), and Uncertainty Accuracy (UA) metrics, as proposed in [36].

The uncertainty (I) of the model is calculated using the following formula:

$$I = H(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) + \sum_c \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^* = c|\mathbf{x}^*, \hat{\boldsymbol{\theta}}_t) \log p(\mathbf{y}^* = c|\mathbf{x}^*, \hat{\boldsymbol{\theta}}_t) \tag{5.6}$$

Here, $H(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D})$ represents the predictive entropy, which measures the amount of information in the predictive distribution. $p(\mathbf{y}^* = c|\mathbf{x}^*, \hat{\boldsymbol{\theta}}_t)$ represents the probability of input $\mathbf{x}^*$ belonging to class c based on the crossbar (weight matrices) $\hat{\boldsymbol{\theta}}_t$ sampled by the arbiter. The formula considers multiple Monte Carlo runs (T) and sums them up to calculate the uncertainty. The uncertainty value (I) is then normalized to obtain $I_{norm} = \frac{I - I_{min}}{I_{max} - I_{min}}$. By applying a threshold I_T on the normalized uncertainty estimation values, the predictions are categorized into certain ($I_{norm} < I_T$) and uncertain ($I_{norm} > I_T$) groups. This categorization leads to four scenarios commonly encountered in Bayesian inference: incorrect-uncertain (iu), correct-uncertain (cu), correct-certain (cc), and incorrect-certain (ic) predictions. The R_{cc}, R_{iu}, and UA metrics can then be calculated based on the number of predictions falling into each of these scenarios, providing insights into the model's performance and uncertainty estimation. The uncertainty estimation metrics are calculated as follows:

R_{cc} This metric measures the probability of a correct prediction given that the model is certain about its prediction. It can be calculated using the formula:

$$P(\text{correct}|\text{certain}) = \frac{P(\text{correct}, \text{certain})}{P(\text{certain})} = \frac{N_{cc}}{N_{cc} + N_{ic}} \tag{5.7}$$

where N_{cc} and N_{ic} represent the counts of correct-certain and incorrect-certain predictions, respectively.

R_{iu} This metric evaluates the probability of the model being uncertain when it makes an incorrect prediction. It can be calculated as:

$$P(\text{uncertain}|\text{incorrect}) = \frac{P(\text{uncertain}, \text{incorrect})}{P(\text{incorrect})} = \frac{N_{iu}}{N_{iu} + N_{ic}} \tag{5.8}$$

where N_{iu} and N_{ic} represent the counts of incorrect-uncertain and incorrect-certain predictions, respectively.

UA This metric represents the overall accuracy of the uncertainty estimation. It is calculated as the ratio of the desired cases (correct-certain and incorrect-uncertain) to all possible cases:

$$\frac{N_{cc} + N_{iu}}{N_{cc} + N_{iu} + N_{cu} + N_{ic}} \tag{5.9}$$

where N_{cu} represents the count of correct-uncertain predictions.

These metrics provide insights into the model's performance and the effectiveness of its uncertainty estimation.

On the other hand, the pixel-wise confidence mask for semantic segmentation is calculated by finding the class-wise sum of the normalized variance of T forward passes. The Intersection Over Union (IoU) metric is the ratio between the area where the predicted and ground-truth masks overlap and their union.

For uncertainty estimation, the Out-of-distribution (OOD) detection capability of the models is evaluated on six datasets that came with PyTorch's vision (TorchVision) framework, e.g.,

1. Gaussian: random Gaussian noise,
2. Noisy $\mathcal{D}_{in}$: random Gaussian noise added to the test set of in-distribution data,
3. SVHN: Google street view house numbers,
4. FakeData: fake data of randomly generated images,
5. STL10: a dataset containing images from the popular ImageNet dataset,
6. and FashionMNIST: grayscale images of 10 clothing classes.

MC-Dropout [35] showed that the prediction probability for OOD data spreads out more compared to $\mathcal{D}_{in}$. Therefore, an OOD is detected if the quantile score of the T forward passes is below 1% and the prediction probability is below 90%. That is to say, an input is rejected if the 99% prediction probabilities for T sub-inferences are below 90%.

5.1.5. Algorithmic Results

In this section, we present predictive performance, uncertainty estimation, and hardware evaluation results of our SpinBayes network on a range of tasks.

5.1.5.1. Predictive Performance

For both semantic segmentation and classification tasks, the experimental results in Table 5.1 (4th column) and 5.2 show that the proposed SpinBayes network performs comparablely to the state-of-the-art (SOTA) methods MC-Dropout [35], MC-Dropconnect [36], and Bayes by Backpropagation (BB-BackProp) [34]. Specifically, relative to the SOTA, performances are generally comparable, with up to 1.14% improvement. This results in up to $\sim$ 114 correct predictions for 10000 validation data. For semantic segmentation tasks, the predicted masks are inaccurate around the boundaries of object classes and obscure classes, such as a cyclist in the CamVid dataset, as shown in Figure 5.9. For our analysis, we have used 2-way ($n = 2$) inference. However, we have evaluated higher and lower n-way inferences. In both cases, the inference accuracies are comparable. Specifically, for $n = 1$, the pixel-wise accuracy of Kvasir biomedical semantic segmentation tasks goes from 94.93% to 94.56%. Similarly, for $n = 3$, the pixel-wise accuracy does not change. In general, we have found that choosing $n > 1$ can improve the accuracy. However, this affects the performance of uncertainty estimates, which will be explored in the following section. Note that because the BNN is a stochastic approach, inference accuracy changes slightly with each evaluation.

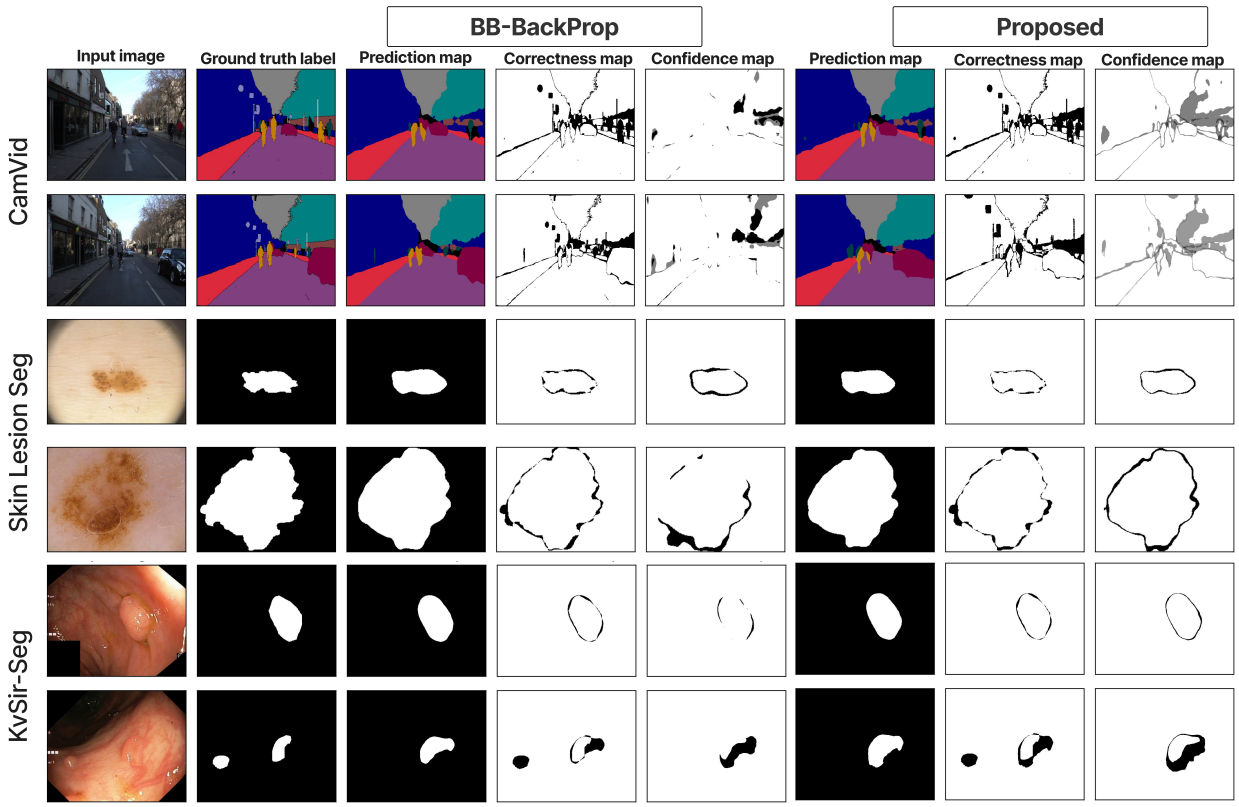


Figure 5.9.: Qualitative analysis of semantic segmentation tasks. The correctness map shows pixel-by-pixel correctness, with white representing correct predictions and black representing incorrect ones. The confidence map shows pixel-by-pixel uncertainty (white regions are correct and certain). **It is recommended to view this figure in color.**

Table 5.1.: Analysis of test prediction error and uncertainty estimation performance for MNIST, CIFAR-10, and CIFAR-100 on popular NN topologies.

Dataset	Method	Topology	Prediction error (%)	Uncertainty metrics (%)		
				Riu	Rcc	UA
MNIST	MC-Dropout[35]	LeNet-5	0.77	31.24	98.77	97.48
	MC-DropConnect[36]		0.57	41.67	99.57	98.87
	BB-BackProp[34]		0.75	94.94	99.95	91.99
	Proposed		**0.77**	**96.81**	**99.96**	**85.10**
CIFAR-10	MC-Dropout[35]	VGG Small	10.57	38.24	92.12	82.89
	MC-DropConnect[36]		10.15	40.29	94.31	87.27
	BB-BackProp[34]		10.17	62.43	95.50	88.22
	Proposed		**10.12**	**82.59**	**97.47**	**83.00**
	BB-BackProp[34]	ResNet-34	7.41	78.40	98.03	82.25
	Proposed		**7.43**	**80.82**	**98.17**	**84.00**
	BB-BackProp[34]	MobileNetV2	9.20	83.86	97.88	78.31
	Proposed		**9.26**	**95.44**	**99.22**	**67.79**
CIFAR-100	BB-BackProp[34]	ResNet-18	30.10	53.50	0.98	53.43
	Proposed		**31.06**	**59.54**	**0.91**	**59.33**
	BB-BackProp[34]	ResNet-34	27.48	46.29	0.86	59.48
	Proposed		**27.91**	**54.22**	**1.05**	**54.17**

Table 5.2.: The quantitative prediction and uncertainty estimate performances of the proposed method and state-of-the-art methods.

Data	Method	Prediction Performance (%)			Uncertainty metrics (%)		
		Pixel acc.	Mean acc.	IOU	Riu	Rcc	UA
CamVid	MC-Dropout[35]	80.99	65.46	47.31	17.23	82.48	80.18
	MC-DropConnect[36]	82.92	67.47	49.53	21.63	86.54	82.78
	BB-BackProp[34]	88.07	73.39	49.17	12.36	92.08	82.17
	Proposed	**89.21**	**74.33**	**52.12**	**27.84**	**99.99**	**92.52**
Skin Lesion	BB-BackProp[34]	96.37	96.37	92.05	56.32	83.26	29.24
	Proposed	**96.16**	**96.16**	**91.53**	**50.46**	**98.95**	**32.46**
KvaSir	BB-BackProp[34]	94.64	94.64	80.39	58.54	90.19	20.55
	Proposed	**94.93**	**94.93**	**81.49**	**48.85**	**99.50**	**20.27**

Table 5.3.: Analysis of the Proposed Method for Detecting Out-of-Distribution Data (OOD).

$\mathcal{D}_{in}$	Topology	OOD Detection (%)					
		Gaussian	Noisy $\mathcal{D}_{in}$	SVHN	FakeData	STL10	FashionMNIST
MNIST	LeNet-5	100.0	99.21	100.0	100.0	100.0	100.0
CIFAR-10	VGG-Small	99.86	94.35	80.33	86.89	89.31	-
	ResNet-34	99.99	98.14	99.99	100.0	99.94	-
	MobileNetV2	98.40	95.31	98.80	97.54	98.06	-
CIFAR-100	ResNet-18	99.99	91.17	98.96	99.45	99.31	-
	ResNet-34	94.45	86.29	68.18	70.01	79.6	-

5.1.5.2. Uncertainty Estimation

In terms of uncertainty estimation, the performance of our approach on the uncertainty estimation matrices R_{iu}, R_{cc}, and UA are significantly better compared to MC-Dropout and DropConnect in the same dataset. The proposed method performs, in general, comparable to the BB-BackProp approach. For example, in the best case, the proposed method improves the uncertainty estimation matrices by 20.16%, as demonstrated in Table 5.1. Furthermore, as shown in Table 5.3, the proposed method can detect up to 100% of the samples from the out-of-distribution dataset.

However, for uncertainty estimation, the 1-way inference ($n = 1$) should be used. Otherwise, more than 40% of the OOD detection capability is reduced. This is because when $n > 1$, the variance for Bayesian inference is reduced due to the local averaging performed at each layer, resulting in a decrease in uncertainty. Please note that since Fashion-MNIST has a single input channel, it is only evaluated on the MNIST model.

Similarly, for semantic segmentation tasks, the proposed method performs significantly better compared to MC-Dropout and DropConnect. Additionally, our results show similar uncertainty estimation matrices compared to BB-Backprop. Specifically, our method can achieve up to 15.69% better uncertainty estimation matrices, as depicted in Table 5.2.

Quantitative observations for semantic segmentation tasks are depicted in Figure 5.9. Ideally, the uncertainty around incorrectly predicted pixels should be higher. That is, the uncertainty should be higher around the black pixels in the correctness mask. Our results show a higher uncertainty around incorrect pixels compared to that of the BB-Backprop approach, indicating that our method produces a better uncertainty mask.

5.1.5.3. Energy Consumption Analysis

The energy consumption of different elements of the proposed architecture is presented in Table 5.4. The AAC, the comparator, and the averaging circuits were synthesized with Synopsys Design Compiler, using

Table 5.4.: Architecture level estimations

Circuit	Dynamic energy	Circuit	Dynamic energy
Memory (Decoding/Sensing)	4.71 pJ	Adder-Accumulator	0.06 pJ
spintronic RNG	3.80 pJ	Comparator	0.01 pJ
Averaging block	18.42 pJ		

TSMC 40 nm PDK. The spintronic Arbiter and the memory operations were evaluated at the circuit level. Furthermore, to assess the energy consumption of a BNN given a LeNet-5 topology, we estimated the number of computational operations required for each layer (e.g., convolution, pooling, and so on). Given the number of operations performed in parallel in the crossbar architecture, along with the dynamic energy evaluation presented in Table 5.4, we utilized a modified NVSim simulator for CiM to scale the architecture and estimate the overall energy consumption for one inference.

In Table 5.6, the energy consumption with 10 forward passes (MC run with $T = 10$) is compared against other FPGA and CiM implementations. For a fair comparison, we used the same metrics and dataset (MNIST) as existing works. The only difference lies in terms of the topology (LeNet-5 CNN topology). We achieve an energy efficiency of 80× better when compared to FPGA implementation [95], 35× better when compared to RRAM [99], and 3× better when compared to an MTJ-based crossbar [102]. Note that the above-referenced implementations involved a smaller network consisting of two linear layers with 200 neurons.

Also, the proposed implementation requires fewer RNGs. For instance, work in [99] encodes all the inputs of the crossbar with a Gaussian RNG based on RRAM variability. Work in [95] implemented a Gaussian RNG that requires a large number of digital circuits (e.g., registers, LUT, etc.). Work in [102] used an array of 4 devices interfaced with an accumulator to implement the Gaussian RNG. Work in [85, 84] also used several stochastic STT-MRAM devices per word-line to implement dropout[35].

For our approach, the total area needed for the implementation of the LeNet-5 topology is 0.508 mm^2, and the delay for 10 forward passes (MC runs with $T = 10$) on an MNIST dataset is 0.512 ms. The study in [85] reports a delay of 2.0 ms, while the study in [95] shows a delay of 0.003 ms. Thus, the delay in the proposed approach is 4× better than that reported in [85]. However, when comparing with [95], the delay for our approach is much higher. This is because the delay reported by the study in [95] utilizes a much simpler topology with only two linear layers, whereas our approach uses the more complex LeNet-5 topology with convolutional layers, which naturally requires more execution time. In a convolutional layer, a small filter/kernel, e.g., a 3×3 kernel, slides over the input data, resulting in more operations and, consequently, longer latency. Also, a pooling layer such as max pooling or average pooling follows one or more convolutional layers in a CNN, which requires additional computations and consequently adds to the latency. Note that the related studies did not report the area associated with their methods. For a fair comparison, we estimated the delay given their topology and for 2 linear layers, and we obtained 0.001 79 ms. Thus, the delay is 1.74× compared to their approach.

As a result, the SpinBayes approach has a much smaller computational overhead associated with the generation of random numbers by using only a single Arbiter with three RNGs for each neural network layer. The proposed approach, in terms of topology and circuit implementation, has the advantage of being much simpler. Thus, only the Averaging block and one Arbiter contribute to the critical path of the circuit in terms of delay and power consumption.

To further evaluate the approach, we implemented the first layer of the topology presented in Table 5.5 with the SpinBayes approach to evaluate the loss in accuracy. Thus, we evaluated 5 different crossbars of a size of 25×6 each. We use the scheme presented in [62] to map the convolutional layer. The traditional mapping consists in unrolling all the kernels in 1D and arranging them in each column of the crossbar array. For the inference, only one writing operation of all weights is required, since they do not change with time. To estimate the loss in accuracy only reading operations are performed. The crossbar takes as

Table 5.5.: LeNet-5 configuration for MNIST dataset

Layer name	Size
Convolution-1	1×6×5×5
Convolution-2	6×16×5×5
Fully Connected-3	3136×120
Fully Connected-4	120×84
Fully Connected-5	84×10

input the MNIST dataset and we evaluate the successful reading operation. An accuracy drop of 0.32% is reported for the first layer compared to the baseline.

Table 5.6.: Comparison of energy consumption with respect to the state of the art

Method	Implementation	Number of RNG	Energy (µJ/Image)
H.Awano et al. [95]	FPGA	- *	21.09
A. Malhotra et al. [99]	RRAM	64*	9.30
S.T Ahmed. et al. [85]	STT-MRAM	20*	2.00
K.Yang et al.[102]	Domain wall-MTJ	Array (2×2)*	0.79
Proposed implementation	**SOT-MRAM**	3	**0.26**

*The number of RNGs increases with the width and depth of the network.

5.1.5.4. Memory Overhead Analysis

Resource requirements are an inherent drawback of most BNNs and ensemble methods. Therefore, the scalability of BNNs to larger topologies is challenging. Since we have used low-bit precision quantization (4-bit), a common model compression technique, our overall memory overhead and consumption are significantly lower.

Indeed, with respect to existing BNN implementations, it requires 8×, 1.6×, and 6.39× less memory compared to the ensemble method [187] with 5 ensembles, MC methods [35, 36], and BB-BackProp [34], respectively, on the topologies we have evaluated (see Table 5.1 and 5.2). We assumed state-of-the-art methods are full-precision 32-bits and require a single cell to store each bit.

Similarly, our proposed approach is scalable for even larger topologies. As can be seen in Figure 5.10, the proposed method achieves the smallest model size for all the CNN topologies. The topologies evaluated here can be found in PyTorch's vision library.

5.1.5.5. Discussion

Here, we discuss the corner cases of our approach.

***n*-ways Inference** In our proposed design, only one crossbar ($n = 1$) is randomly selected for an n-way inference. First, selecting one crossbar reduces run-time power consumption since only one crossbar array is used at a time. Additionally, activating and accessing multiple crossbars simultaneously increases periphery overhead and complexity. Moreover, this choice is also motivated algorithmically and by the Bayesian paradigm. Although ($n > 1$) slightly increases accuracy, the reduction in OOD detection capabilities is significant. Therefore, $n = 1$ is recommended as the main goal of our approach is to provide low-cost uncertainty estimates without sacrificing performance.

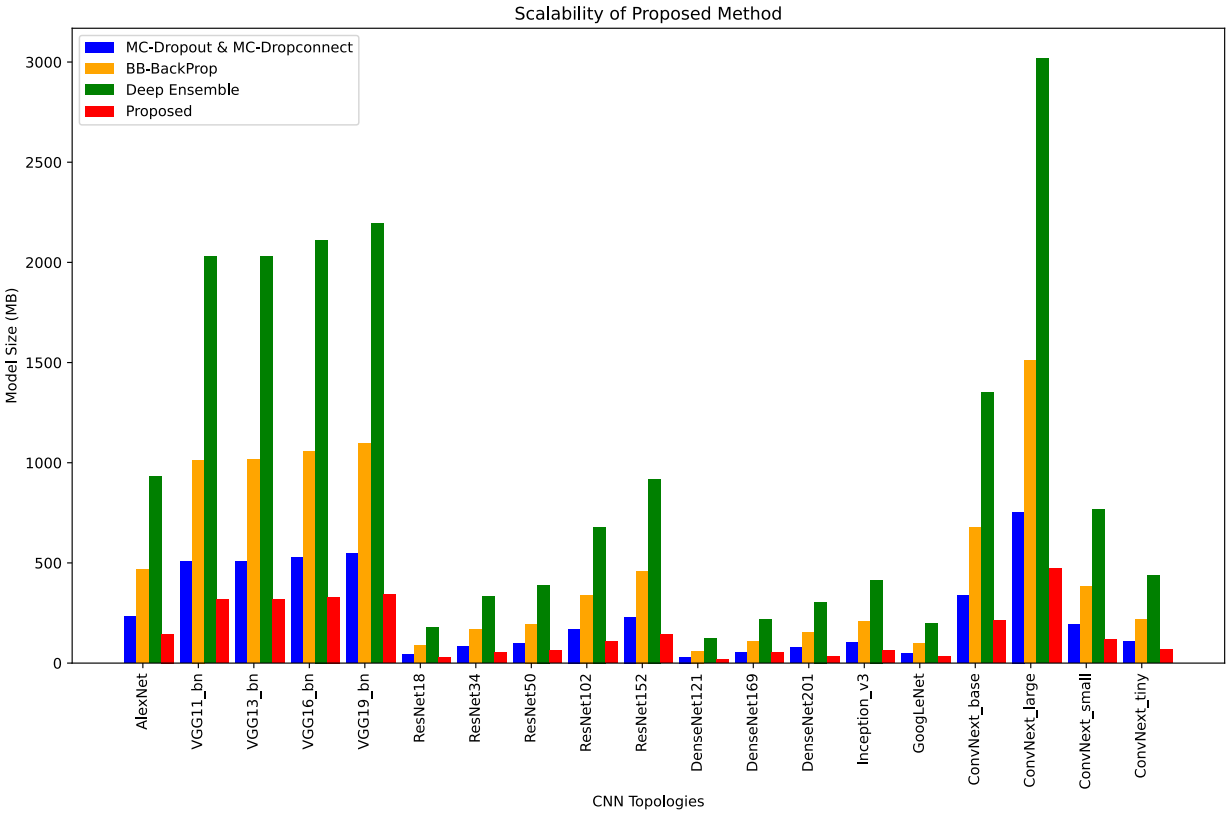


Figure 5.10.: The scalability of the proposed method is evaluated in terms of model size (in MB) for different large CNN topologies. The model size for each of the topologies is compared to SOTA BNN methods, namely MC-Dropout [35] & MC-Dropconnect [36], BB-BackProp [34], and Deep Ensemble [187]. The proposed method achieves a significantly smaller model size compared to the other BNN methods across all CNN topologies. **It is recommended to view this figure in color.**

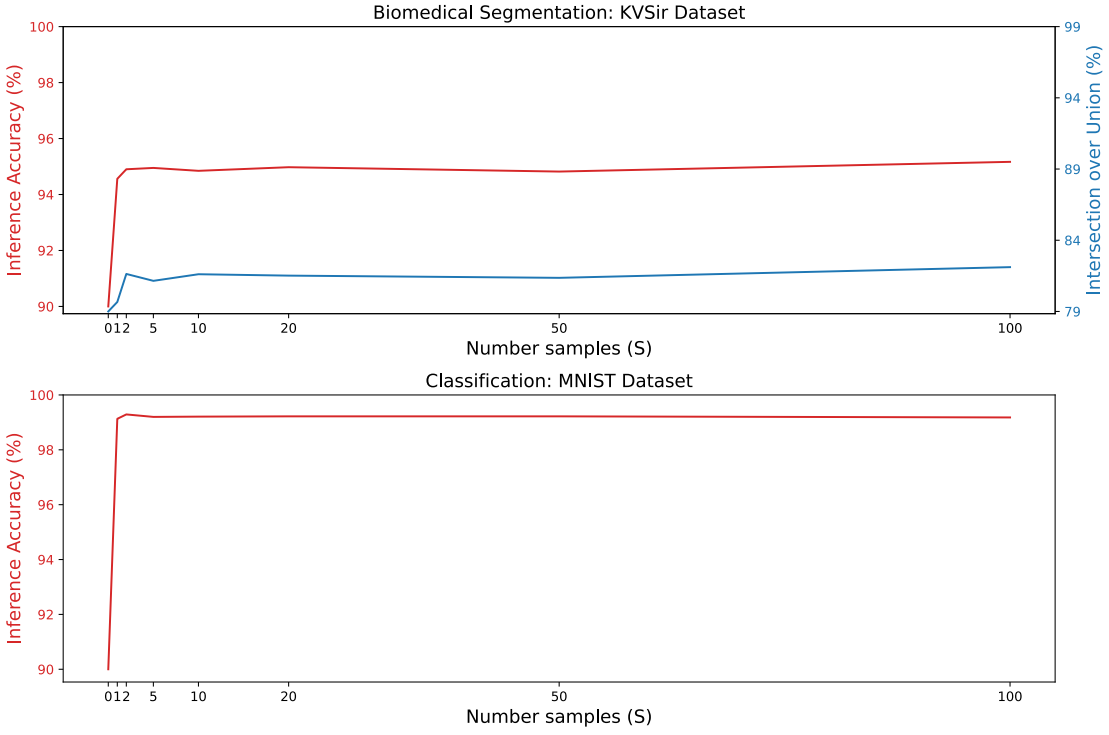


Figure 5.11.: The relationship between inference accuracy and the number of samples (S) for biomedical segmentation task on the KVSir dataset (top) and classification task on the MNIST dataset (bottom). **It is recommended to view this figure in color.**

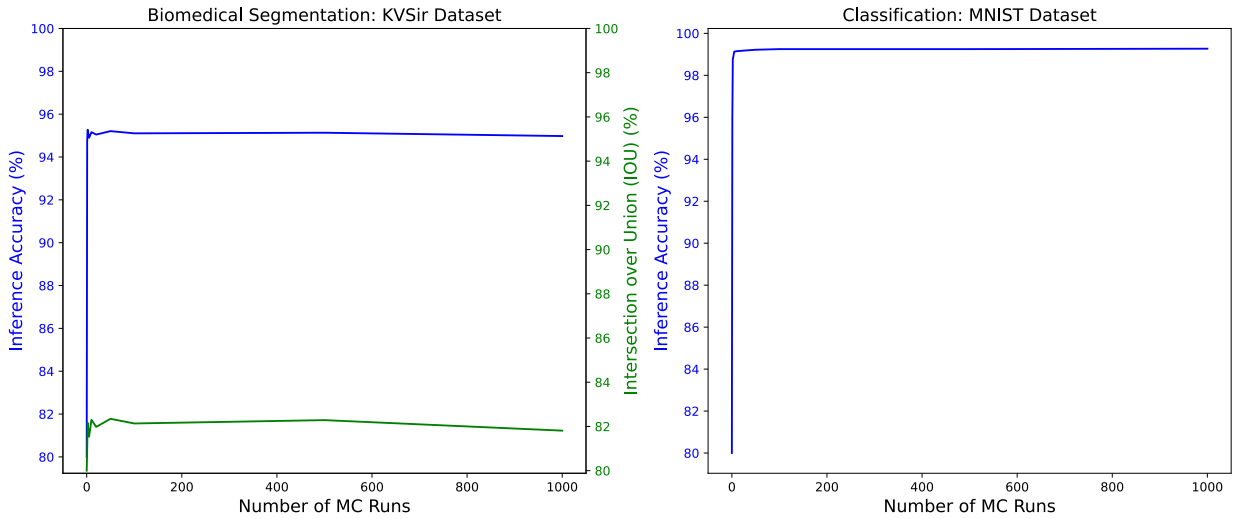


Figure 5.12.: Inference accuracy for different numbers of Monte Carlo (MC) runs for biomedical image semantic segmentation (left) on the KVSir dataset and image classification (right) on the MNIST dataset. **It is recommended to view this figure in color.**

Model Compression When it comes to choosing the bit precision for the parameters, there is a trade-off between accuracy and hardware overhead. A higher bit precision for the parameters would improve accuracy, but implementing a high bit precision would require challenging sensing schemes and result in hardware overhead due to complex ADCs. Furthermore, most non-volatile memory devices achieve only a few stable conductance levels [200]. Thus, for an efficient CiM implementation, sufficiently lower bit quantization is required. In our analysis and circuit design, we have chosen 4-bit precision to represent the parameters, as it offers the best trade-off between accuracy and hardware overhead.

Although we have shown that our method has significantly smaller memory requirements, if further compression is required, unimportant neurons can be removed through pruning [201]. However, such a method usually requires sensitivity analysis. Additionally, since the size of the memory array is fixed, pruning may not be beneficial as it can lead to underutilized memory cells. CiM architecture-aware pruning can be performed [202], but evaluating such an approach is beyond the scope of this paper.

Number of Samples Required for the Bayesian In-Memory Approximation The number of samples J taken for the proposed *In-Memory Approximation* directly reflects the number of crossbars per layer required. Although in our case, we have taken only five samples ($J = 5$), we have found that taking more samples is not beneficial in terms of performance, as shown in Figure 5.11. As the number of samples increases, the accuracy reaches saturation beyond a certain number of samples ($J = 2$ for both MNIST and Kvasir datasets). A further increase in the number of samples only leads to marginal improvements in accuracy (and Intersection over Unions (IoUs) for the semantic segmentation task) but incurs significant hardware overhead. This suggests the possibility of reducing the number of spintronics crossbar arrays required to implement the proposed SpinBayes network. Therefore, this further underscores the efficiency and scalability of the proposed *In-Memory Approximation* method for cost-effective BNN implementation with large-scale neural network models.

Reducing the number of samples per layer reduces the number of crossbars and also the complexity of the circuit. Indeed, with the increase in crossbars, the required periphery must be adapted to handle the implementation of multiple crossbars. Thus, for $J = 2$, only one RNG is required for two crossbars, allowing energy, power, and area savings.

Compatibility with CMOS fabric and other non-volatile memories The proposed approach investigates the benefits and challenges of spintronic technology in the context of implementing BNNs on CiM-based

hardware accelerators. The specific advantages offered by spintronic technologies, such as their memory-like behavior and stochastic characteristics, are crucial for Bayesian inference. However, it is important to note that the proposed SpinBayes topology is not limited to spintronic implementations alone. It can also be implemented using alternative technologies, such as non-volatile memory solutions [200], or even through a fully CMOS-based approach for CiM architecture. The main considerations for such implementation are the addition of the stochastic arbiter and the utilization of an Averaging block in case $n > 1$.

Number of Monte-Carlo Runs for the Bayesian Inference The number of Monte Carlo runs (forward passes) required for a BNN inference can significantly increase runtime power consumption and latency per inference result. We have found that increasing the number of Monte Carlo runs beyond a certain point does not increase accuracy (as shown in Figure 5.12), but only increases runtime power and latency. In both the cases of biomedical semantic segmentation and classification tasks, the accuracy of the proposed SpinBayes network saturates after a small number of Monte Carlo runs, e.g., 5 for the MNIST dataset and 2 for the KvSir dataset, respectively. The proposed SpinBayes network is effective in achieving high accuracy with a small number of Monte Carlo runs, which can significantly reduce the computational cost of inference. In comparison, as reported in the work by Mobiny et al. [36], the MC-Dropconnect approach achieves an inference accuracy that is within one standard deviation from its best performance (obtained at 90 Monte Carlo runs) after 18 Monte Carlo runs, while for the MC-Dropout method, 54 Monte Carlo runs are required to reach its best performance (obtained at 94 Monte Carlo runs).

5.1.6. Scientific Impact of This Work

The scientific impact and contributions of this work can be summarized as follows:

1. **AI Accelerator Centric Approximation:** We introduce a novel Bayesian in-memory approximation method that enables the efficient mapping of learned probabilistic distributions of BayNNs to spintronic-based CiM architectures. Our approach significantly reduces the memory overhead and energy consumption compared to traditional methods. Therefore, our work allows more practical VI-based BayNN implementations for real-world applications. Therefore, research in the direction of AI accelerator-centric BayNN approximation should be considered for highly efficient BayNN implementation.
2. **Novel AI Accelerator Suitable BayNN Topology:** The proposed SpinBayes network topology leverages the stochastic and deterministic properties of Spintronic devices for efficient on-the-fly sampling during Bayesian inference. Consequently, the proposed topology reduces the number of required RNGs and requires a fixed number of RNGs irrespective of the size of the topology. Thereby lowering power consumption and enhancing the scalability of BNN implementations on edge AI hardware accelerators. Consequently, research in the direction of an AI accelerator with suitable BayNN topology should be considered.
3. **Scalability:** Our proposed topology is also scalable for other AI accelerator architectures and various NN topologies, including existing CNN, RNN, and MLP topologies.
4. **Energy and Performance Efficiency:** The proposed implementation of BayNN spintronic-based CiM architecture results in a substantial reduction in energy consumption and computational latency. Compared to state-of-the-art implementations, the SpinBayes network achieves significantly lower energy consumption and significant improvements in latency. Therefore, making it highly suitable for resource-constrained edge AI accelerators.
5. **Post Training Quantization For Efficient Resource Utilization:** Our work also proposes a post-training quantization method that allows overcoming the limited stable states of memory cells and reduces mismatches between parameter bit-width and memory cells. The proposed method can be modified for other bit-with requirements of an edge AI accelerator. Therefore, it ensures that the model parameters fit the constraints of memory cells in an edge AI accelerator.

6. **BayNN for Real-time Applications:** The overall reduction in energy consumption and improvement in computational efficiency make the proposed method highly suitable for real-time applications where low latency and highly reliable predictions are critical.

5.1.7. Section Conclusion

In this section, we propose SpinBayes, a Bayesian inference approach suitable for multi-value Spin-Orbit Torque-based Computing-in-Memory hardware accelerators, from a holistic perspective. We propose a multilevel SOT-based bitcell to map quantized parameters for Bayesian inference. Additionally, we develop a spintronics-based Arbiter that utilizes the inherent stochasticity of SOT devices to randomly select crossbar arrays for each forward pass. Our proposed architecture has been rigorously examined on various classification and semantic segmentation tasks for prediction performance and uncertainty quantification. It enhances prediction performance by up to approximately 1%, can detect up to 100% of various out-of-distribution data, and generates superior uncertainty masks for semantic segmentation tasks. Furthermore, it exhibits a memory overhead that is 8× smaller and energy consumption that is 80× lower compared to state-of-the-art CMOS implementations.

5.2. Bayesian Subset Parameter Inference

In our previous work, we overcome the challenges of mapping the variational distribution and sampling it on the fly during Bayesian inference with several contributions. However, there are still some research questions and challenges: (a) Is it possible to implement variational Inference in binary NNs? It can effectively overcome the limited stable states of the Spintronics devices and allow the reuse of existing crossbar arrays. (b) Can memory consumption and power be further reduced for the extreme edge AI applications? Although we showed that our previous work significantly reduced the memory consumption, the resources in extreme edge AI devices are severely limited. (c) Can other parameter groups in an NN other than weights or activation be treated as Bayesian?

This work aims to address these research questions and is based on our publication at the IEEE DATE 2023 [49].

5.2.1. Observation and Motivation

In NN, the weight matrices of the fully connected and convolutional layers consume the most storage memory, as they are utilized to calculate the weighted sum of the input. For example, in ResNet-18 topology, weight matrices consume 98.6% of total parameters while biases 0.3% and other parameters consume 1.05%. In Bayesian NNs, since the variational distribution $q_{\omega}(\theta)$ with parameters ω is applied to the weight matrices, memory consumption increases significantly.

The higher computational complexity comes from the need for sampling from $q_{\omega}(\theta)$ during Bayesian inference. In addition, as stated in the previous work, the implementation of the variational distribution $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$ with CiM-based hardware accelerators can be challenging and may require changes to the normal memory structure. Furthermore, because of the limited stable states of Spintronics devices, conventional hardware implementations may not be efficient. This is because the mapping of parameters of the distribution, e.g., mean and variance, to a crossbar array, may change the overall distribution due to quantization error. Consequently, the distributions can differ considerably from the trained model. In earlier work [104], a novel network topology, crossbar structure, multi-bit memory cells, and a Bayesian approximation method were developed to overcome the challenges mentioned above. Here, we take another approach. Our approach allows for *Binary Bayesian neural networks* with the variational inference approximation with significantly lower memory consumption.

5.2.2. Bayesian Subset Parameter Inference

In this work, we propose an efficient Bayesian NN framework with both deterministic and stochastic parameters. In particular, the larger group of parameters, such as the weights and biases of linear and convolutional layers, are kept deterministic. This allows us to implement variational inference in Bayesian binary neural networks. On the other hand, the scale parameters of binary NN, which is one of the smallest groups of parameters, are treated as a random variable with a probability distribution. Consequently, this allows a) direct implementation to Spintronis-based crossbar arrays, b) to overcome the limited stable state challenge of Spintronics memory, c) reuse the existing crossbar structure, and d) to reduce memory overhead by 32×. As a result, the memory and computational complexity are drastically reduced compared to conventional BayNNs, and the adaptation is CiM-hardware-friendly.

For our approach, we consider the following approximation of the signature weighted sum computation of an input vector $\mathbf{z}^{(l-1)}$ with a weight matrix $\mathbf{W}^{(l)}$ of a layer l as

$$\mathbf{z}^{(l-1)}\mathbf{W}^{(l)} \approx \text{sign}(\mathbf{z}^{(l-1)})\,\text{sign}(\mathbf{W}^{(l)}) \odot \boldsymbol{\alpha}^{(l)}, \tag{5.10}$$

where $\boldsymbol{\alpha}$ is a vector of learnable parameters representing *scale* and $\text{sign}(\cdot)$ is to be considered elementwise. Since we consider Bayesian binary neural networks, the activations $\mathbf{x}$ and weights $\mathbf{W}$ are binarized $\{+1, -1\}$ with the sign function. In contrast, the entries of the scaling vector $\boldsymbol{\alpha}$ are typically considered 32-bit (float) values, but we apply a further approximation for the CiM implementation.

For learning, a distinct treatment is applied to the two-parameter groups. Specifically, we apply a Bayesian treatment to learning $\boldsymbol{\alpha}$ via variational inference and learn a distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\alpha})$, while the rest of the parameters, which we denote by $\boldsymbol{\theta}$ in the following (e.g., the weights $\mathbf{W}$ for each layer), are learned via a (classical) maximum likelihood approach. The overall training objective is defined as

$$\max_{\boldsymbol{\theta},\boldsymbol{\omega}} \quad p(\mathcal{D} \mid \boldsymbol{\theta}) - \lambda \cdot \text{KL}\left(q_{\boldsymbol{\omega}}(\boldsymbol{\alpha}) \,\|\, p(\boldsymbol{\alpha} \mid \mathcal{D})\right), \tag{5.11}$$

where λ denotes a hyper-parameter that is to be set. Note that this objective cannot be directly optimized since the KL term is intractable and therefore replaced with the evidence lower bound (ELBO) approximation, which provides a lower bound on the KL [203].

Due to the hardware constraints, we consider several approximations. Specifically, the parameters $\boldsymbol{\theta}$ need to be binarized, while samples from $q_{\boldsymbol{\omega}}(\boldsymbol{\alpha})$ are also quantized to low bit-precision. To enable gradient-based learning, quantizations are only considered in forward passes (during training and inference), whereas they are disregarded while computing gradients. This is generally known as the straight-through (gradient) estimator [44].

To efficiently implement the Bayesian NN in a CiM-architecture, we take a set $\mathcal{S} = \{\boldsymbol{\alpha}^{(n)} \sim q_{\boldsymbol{\omega}}(\boldsymbol{\alpha}) \mid j = 1, \cdots, J\}$ of J samples from $q_{\boldsymbol{\omega}}(\boldsymbol{\alpha})$. These samples are then mapped to a specific crossbar array. During the online operation, a stochastic sampler is used to sample one of the crossbars in each forward pass. Consequently, the distribution $q(\boldsymbol{\alpha}) = \text{Choose}(\mathcal{S})$ that selects samples from $\mathcal{S}$ uniformly, approximates the distribution of the (quantized) samples from the variational distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\alpha})$.

Hence, through the CiM-hardware, the distribution of $\mathbf{y}^*$ given $\mathbf{x}^*$ is approximated as

$$\begin{aligned} p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) &\approx \frac{1}{T}\sum_t p\left(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}, \boldsymbol{\alpha}^{(t)}\right) \\ &\text{with} \quad \boldsymbol{\alpha}^{(t)} \sim q(\boldsymbol{\alpha}) = \text{Choose}(\mathcal{S}). \end{aligned} \tag{5.12}$$

Note that the parameters $\boldsymbol{\theta}$ (e.g., weights) are considered deterministic, while the learned distribution of the scales $\boldsymbol{\alpha}$ is used to express the uncertainty in the predictions.

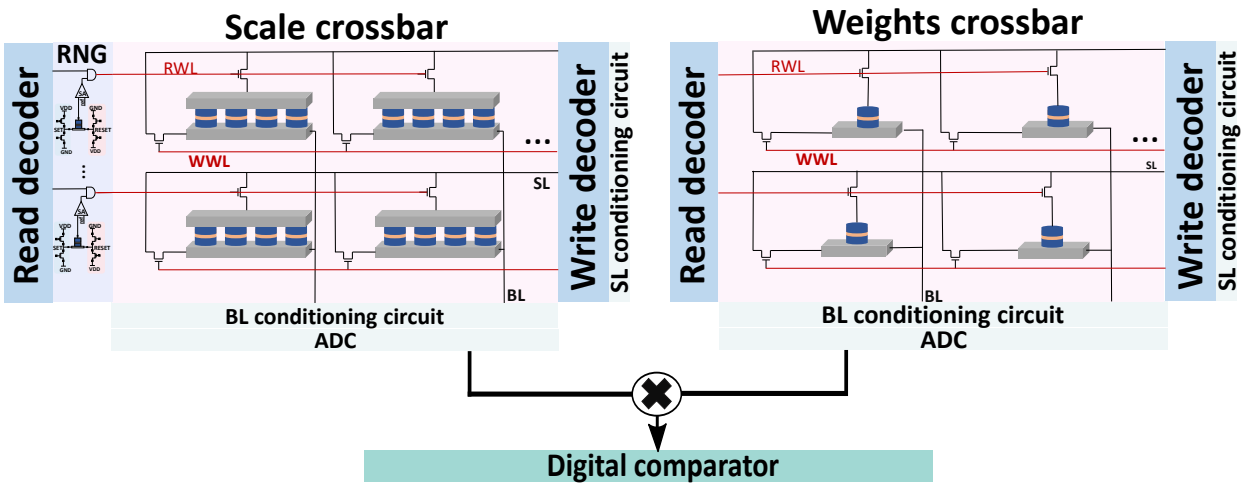
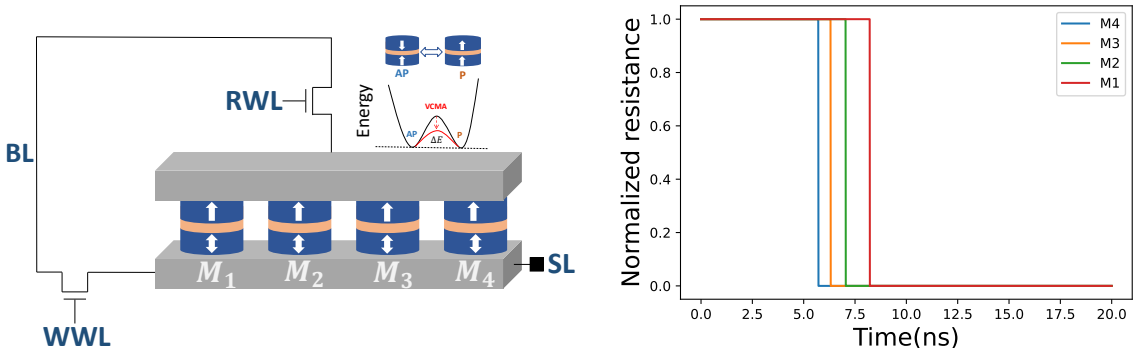


Figure 5.13.: Proposed spintronic architecture.



(a) Multi-value cells with 4 MTJs

(b) Multi-value cell switching

Figure 5.14.: Proposed multi-value SOT bit-cell. **It is recommended to view this figure in color.**

5.2.3. Hardware implementation

5.2.3.1. Bayesian Inference Architecture

To implement the proposed Bayesian inference, a new CiM architecture is proposed according to the functionality of equation equation 5.10. The architecture consists of two crossbars per layer, one maps the Bayesian scale, and the other the weights. The scale crossbar is implemented using a multi-level device that stores the quantized parameters. In addition, a stochastic spintronics device is used to allow the random selection of the different scales in each forward pass based on the equation equation 5.12. Each crossbar is equipped with two decoders, one for reading and the other for writing. Decoders allow for the selection of multiple devices for reading and writing operations. The stochastic sampler is connected to the different signals of the reading decoder of the scale crossbar. The MVM operation in a crossbar array is achieved by activating multiple wordlines in parallel. At the output of the crossbar, the current is sensed and converted to a digital signal with a flash ADC. As shown in Figure 5.13, we multiply the results of the in-memory computation on a crossbar by the randomly selected scale and apply a sign function with comparators in each layer. The model predictions are accumulated for T Monte-Carlo runs with the Adder ACcumulator (AAC) module.

5.2.3.2. Scale crossbar

To quantitatively represent the parameters required for our proposed BayNN approach, we have implemented a multi-level device composed of multiple MTJs placed on a single SOT track. We used four MTJs to ensure reliable reading and writing of the cell. Each multi-level device is able to store up to five levels of conductance (e.g., 4AP, 3AP-1P, 2AP-2P, 1AP-3P, 4P). To achieve a wider range of conductance levels and parameter representation, several of these multi-value cells can be used in tandem. As mentioned earlier, this device exploits both the VCMA and SOT effects for the writing process. The parallel MTJs share the same top and bottom electrodes. Thus, only two access transistors are needed for read and write operations, as in the conventional SOT device, see Figure 5.14. Consequently, the cell becomes denser, and hardware resource efficiency improves. The access transistors are controlled by a Write Word-Line (WWL) and a Read Word-Line (RWL). At a given SOT current, the RWL signal can vary to program individual MTJs or multiple MTJs at a time.

5.2.3.3. Weights crossbar

For the crossbar that encodes weights, we decided to implement binary synapses with SOT technology, where the resistance state of the device (R_{P} and R_{AP}) represents the binary value (+1, −1). The current sum due to the activation of multiple cells will be compared to a reference. The reference serves as an activation function and batch normalization via approximation proposed [204, 205] for the deterministic crossbar array.

5.2.3.4. Stochastic SOT device

By utilizing the stochastic behavior of SOT devices as a random number generator (RNG), we were able to introduce the desired random sampling. Two CMOS drivers were used to generate the bidirectional current across the heavy metal to switch between R_P and R_{AP} states. To attain a given probability with the stochastic device, "SET" and "RESET" operations were repeated to program the MTJ. After a "SET" operation, the MTJ is sensed to evaluate its state, and then "RESET". As a result, the successive "SET" and "RESET" operations generate a stochastic bitstream.

5.2.4. Experimental Result

5.2.4.1. Simulation Setup

The predictive performance of the proposed method is evaluated on MNIST, Fashion-MINST, and CIFAR-10 benchmark in-distribution datasets on MLP, LeNet, and VGG based on [36] topologies. Weights and activation BayNNs are binarized according to [47] and the proposed Bayesian scale is quantized to 4-bit using the algorithm proposed in [173]. A value of 0.001 is used for the hyperparameter λ in Equation 5.11.

5.2.4.2. Predictive Performance

For MLP on MNIST, our results depicted in Table 5.7 show that the proposed subset parameter inference performs comparably to SOTA full-precision Bayesian methods with only a 0.51% difference in accuracy. Similarly, inference accuracy is comparable, i.e., within 0.68% of full-precision and binary point estimate NNs. Generally, point estimate methods outperform Bayesian methods. Therefore, the difference is slightly greater in comparison.

Table 5.7.: Predictive Performance of MNIST dataset on four-layer MLP in comparison to related Bayesian and point estimate methods. The superscript * represents methods that are point estimates.

Method	Bit-width (W/A)	Inference Accuracy
FP (ReLU)*	32/32	98.78%
FP (Tanh)*	32/32	98.39%
MC-Dropout	32/32	98.61%
IR-Net*	1/1	98.26%
Proposed	1/1	**98.10%**

Table 5.8.: Illustration of the proposed method's prediction performance on the Fashion-MNIST dataset using the LeNet-5 CNN topology, compared to Bayesian and point estimate approaches with varied bit-widths of weights and activation (W/A). The superscript * denotes point estimates methods.

Method	Bit-width (W/A)	Inference Accuracy
FP (ReLU)*	32/32	92.01%
FP (Tanh)*	32/32	91.78%
MC-Dropout	32/32	91.71%
Deep Ensemble	32/32	91.68%
IR-Net*	1/1	91.71%
Proposed	1/1	**92.0%**

Furthermore, the inference accuracy of the proposed method on CNN topologies on Fashion-MNIST and CIFAR-10 is still comparable to SOTA Bayesian methods as depicted in Tables 5.8 and 5.9. Specifically, the inference accuracy of Fashion-MNIST is 0.01% and CIFAR-10 is 2.54% lower in the worst case. Since CIFAR-10 is a much harder dataset, the difference is larger for our proposed method. However, when our proposed method is compared with SOTA binary method, our method slightly can improve the accuracy, e.g., by 0.62% for VGG. Due to the fact that binary activation is an approximation of Tanh activation, the performance of our proposed method is closer to a full-precision model with Tanh activation, but with ReLU activation, the difference is slightly greater.

5.2.4.3. Uncertainty Estimation

To reiterate, typically, we assume that the distributions of training and test data are identical. However, when the distributions of training and test data differ, e.g., when the test data is rotated or corrupted with noise, we expect the uncertainty of the model or of the prediction to be high. We have performed two experiments with varying intensity for dataset shift. In one case, we continuously rotated the image by 7 degrees in 12 steps, and in the other case, we added random uniform noise with increasing intensity. It can be seen in Figure 5.15 that the inference accuracy decreases, and the negative log-likelihood (NLL) increases. NLL is a standard method for estimating uncertainty, and a well-trained model typically has a low NLL score. A higher NLL score than a predefined threshold, e.g., mean NLL on test data, indicates OOD data. We can detect up to 64.34% of OOD data with this approach.

5.2.4.4. Analysis of Hardware Implementation

The energy consumption of the proposed architecture is presented in Table 5.10 and is compared to related works. The multiplier, the AAC block, and the comparators were synthesized with Synopsys Design Compiler using the TSMC 40nm PDK. The crossbar and the stochastic device were then simulated in Spice. Furthermore, the design was scaled up using the CiM version of NVSIM, and we evaluated the energy consumption on a LeNet-5 and small VGG topology. For each topology, we performed T=10 forward passes on the MNIST dataset. The energy consumption of 0.30 µJ is reported with the LeNet-5 topology and 2.00 µJ with the small VGG topology (9-layers). The proposed architecture is 70× more energy efficient

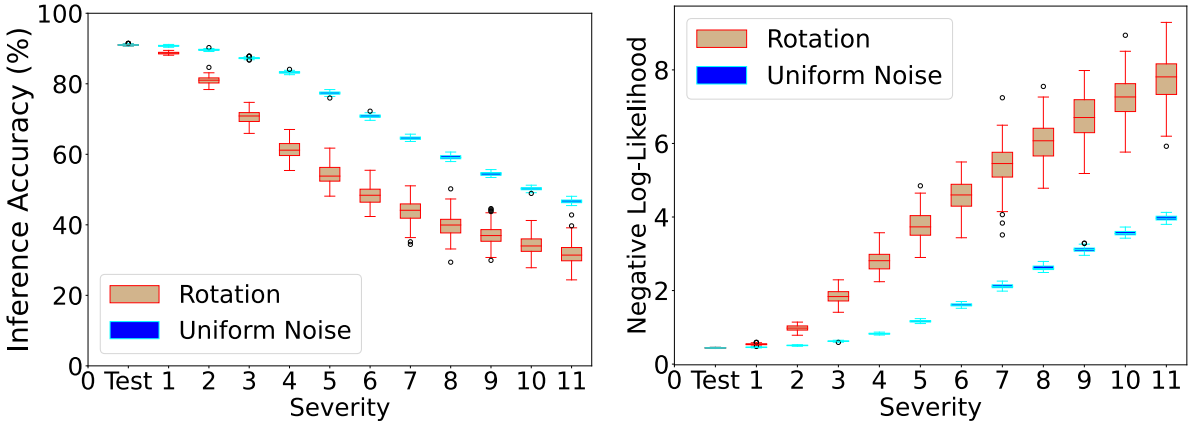


Figure 5.15.: Evaluation of out-of-distribution performance on Fashion-MNIST dataset. The images are rotated by 7°, and uniform noise is added to shift the distribution. **It is recommended to view this figure in color.**

Table 5.9.: Prediction performance of our method is compared to Bayesian and point estimate approaches utilizing the CIFAR-10 dataset and different bit-widths of weights and activation (W/A). * denotes point estimation methods.

Topology	Method	Bit-width (W/A)	Inference Accuracy
VGG	FP (Tanh)*	32/32	91.23%
	FP (ReLU)*	32/32	93.31%
	MC-Dropout	32/32	92.79%
	IR-Net*	1/1	89.96%
	Proposed	**1/1**	**90.62%**
ResNet-18	FP (Tanh)*	32/32	91.33%
	FP (ReLU)*	32/32	93.77%
	MC-Dropout	32/32	93.44%
	IR-Net*	1/1	91.5%
	Proposed	**1/1**	**90.5%**

when compared with FPGA implementation [95], 31× better when compared to RRAM [99], and 2.63× better when compared to an MTJ-based crossbar [102]. All studies were evaluated on the MNIST dataset but on different topologies, LeNet-5 in our case, while other studies were only evaluated on two linear layers.

In terms of memory consumption, compared to SOTA methods for uncertainty estimation, our proposed method requires 63.49× lower storage memory compared to variation inference approximation [34], 158.78× lower storage memory compared to the ensemble approach (with 5 ensembles) [187], and 31.76× lower storage memory compared to Dropout-based [35] approximation. Furthermore, even compared to 1-bit binary NNs with point estimate parameters, our proposed BayNN requires ~ 1% lower storage. Since we quantize the scale, our method is even lower than SOTA binary NNs which have 32-bit scales. Furthermore, we assumed that one-bit cell is required for each bit of parameter storage. The variables of the model are not taken into account.

Table 5.10.: Energy comparison with SOTA implementation

Method	Implementation	Energy (μJ/Image)
H.Awano et al. [95]	FPGA	21.09
A. Malhotra et al. [99]	RRAM	9.30
K.Yang et al.[102]	Domain wall-MTJ	0.79
Proposed implementation	**SOT-MRAM**	**0.30**

5.2.5. Scientific Impact of This Work

The scientific impact and contributions of this work can be summarized as follows:

1. **Subset-Parameter Inference:** This work introduces a novel approach, where only the smallest subset of a network parameters group are defined as probabilistic, while the rest are kept deterministic. The subset parameter is strategically selected for low-cost Bayesian inference. As a result, the overhead associated with Bayesian inference in the CiM architecture is drastically reduced without compromising the quality of uncertainty estimates.
2. **Resource-Efficient Variational Inference:** Our proposed approach enhances the feasibility of deploying complex Bayesian neural networks with variational inference in resource-constrained edge devices.
3. **Efficient Hardware Utilization:** Our proposed approach allows variational inference in binary neural networks. Consequently, existing CiM architectures, without needing significant modifications, can be utilized for Bayesian inference.
4. **Faster Sampling From Posterior Distribution:** We propose a design space exploration that enables low-cost and fast sampling from the posterior distribution for Bayesian inference. Specifically, the inherent stochastic behavior of SOT-MRAM devices is utilized.
5. **Robust Uncertainty Estimation:** Our work paves the way for reliable and hardware-efficient AI in critical applications in an uncertain or dynamic environment.

5.2.6. Section Conclusion

In this section, we present a low-cost and scalable Bayesian neural network framework suitable for CiM hardware. Our method deals with larger groups of parameters in a deterministic method, and Bayesian processing is only applied to a specific group of parameters, *scale*. A novel CiM architecture with two separate crossbars per layer is presented for the Bayesian inference. One crossbar stores deterministic weights, while the second array stores the Bayesian scale. A multilevel SOT-based bitcell is designed to map quantized Bayesian scale parameters. Furthermore, the stochastic behavior of the MTJ is harnessed to implement sampling from the posterior distribution of the variational distribution. Our proposed Bayesian NN is rigorously examined for its prediction performance and uncertainty quantification. We show that the prediction performance is comparable to SOTA methods with different bit widths. Furthermore, the energy consumption and memory requirement were evaluated on large topologies. Compared to the SOTA Bayesian implementation, the energy consumption is 70× smaller than that of a CMOS-based implementation and 31× smaller than that of an RRAM-based implementation. The Storage Memory requirement is up to 158.78× lower.

6. Model Ensemble-Based Uncertainty Estimation

This thesis made several contributions in the area of variational inference and Monte Carlo-Dropout as a Bayesian approximation. Another alternative option for uncertainty estimation is the model ensemble approach. The model ensemble, such as the Deep Ensemble [187], is considered a "gold standard" for uncertainty estimation [206]. In the Deep Ensemble, M ensemble members $1, \cdots, M$ are trained independently and stored in hardware. During inference in edge AI accelerators, each model processes the input in M forward passes (see Fig. 6.1 (a)). Subsequently, the outputs of all models are combined to obtain the predictive distribution. Therefore, the cost in terms of latency, power, and memory for training, storage, and processing of M ensemble members is challenging for edge AI accelerators with limited resources.

Therefore, this work aims to address challenges by proposing the Tiny Deep Ensemble approach, a low-cost ensemble method for uncertainty estimation in deep neural networks. The work is based on our paper [115].

6.1. Motivation and Observation

We observed that parameters other than weights and biases in an NN consume only $\sim 1-2\%$ of all parameters, as shown in Fig. 6.2. Therefore, we propose the Tiny-Deep Ensemble (Tiny-DE) method, in which only normalization layers are ensembled M times, with all ensemble members sharing common weights and biases. Recall that the normalization layer is commonly used in NNs, as it speeds up training and improves performance [50]. Our approach is scalable 1) in any AI accelerator architecture, 2) in any

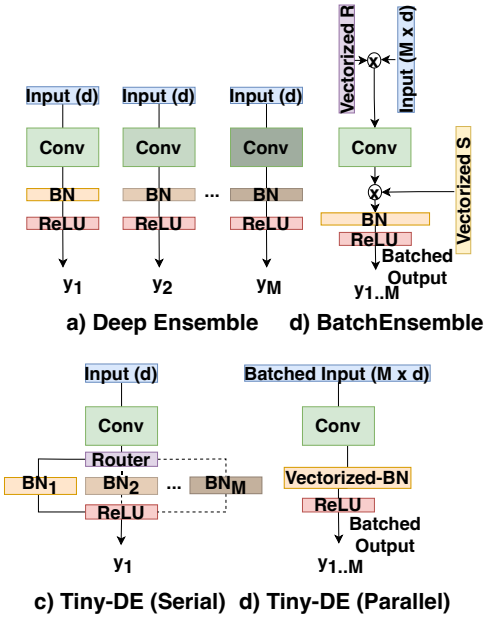


Figure 6.1.: a) Deep Ensemble [187] with M ensemble members , b) BatchEnsemble [114], proposed Tiny-DE model with M normalization layers with *a single shared convolutional layer* in c) serial mode, and d) parallel mode.

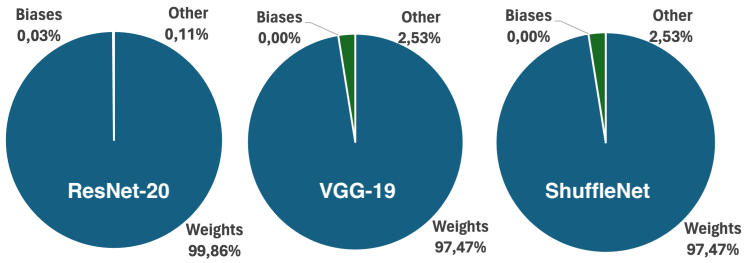


Figure 6.2.: Share of parameter groups with respect to the total number of parameters in different CNN topologies.

NN topologies, such as CNNs and recurrent neural networks (RNNs), 3) in tasks, and 4) in datasets. Furthermore, our approach is parallelizable during training and inference within an AI accelerator architecture. Consequently, all ensemble members can be updated concurrently for a given mini-batch, and inference requires a single forward pass, allowing for *single-shot training and inference*.

6.2. Methodology

6.2.1. Core Idea

In Tiny-DE, only the normalization layers are ensembled, which overall have the smallest amount of parameters in the network, while all other parameters are shared. We denote the normalization layers of a layer index l by $\text{BatchNorm}^l_{\gamma_0,\beta_0}, \ldots, \text{BatchNorm}^l_{\gamma_{M-1},\beta_{M-1}}$ in the following. in the following. The normalization layers can be Batch Normalization, Layer Normalization, Instance Normalization, and Group Normalization with learnable parameters $\beta \in \mathbb{R}^{C_{out}}$ and $\gamma \in \mathbb{R}^{C}_{out}$. Therefore, compared to the deep ensemble approach [187] and BatchEnsemble [114], our approach requires a $M\times$ lower weight matrix storage and a $2M\times$ lower rank-1 matrix computation (see Figs.6.1 (a) and (b)).

6.2.2. Operation Modes

Depending on the batch processing capabilities of the hardware architecture, Tiny-DE can operate in either sequential or parallel modes. In hardware architectures where batch processing is challenging, such as the memristor-based CiM architecture [207, 208, 209], a sequential processing NN architecture should be used. Here, "sequential" refers to sequential in time rather than signal flow through the ensembles. In contrast, in parallel mode, *single-shot uncertainty estimation* can be done using vectorization in hardware architectures such as edge tensor processing units (TPUs), field-programmable gate arrays (FPGAs), and graphics processing units (GPUs) [16, 210]. Both methods are described in detail in the following.

6.2.2.1. Sequential Inference

The sequential inference of Tiny-DE utilizes a counter variable c and router to dynamically select a normalization layer for each forward pass. Depending on the state of the counter c, the output of the l-th layer $\mathbf{z}^l$ is directed through one of the M normalization layers, as shown in Fig. 6.1 c). The activation function, such as the ReLU function, is applied to the processed output as is normally done.

The counter c is an unsigned integer and it is updated cyclically in each layer as follows:

$$c \leftarrow (c+1) \mod M, \tag{6.1}$$

where c is initialized to 0 at the start of the inference process. The mechanism ensures that the output of each layer sequentially passes through each normalization layer in a cyclic order. For example, if $c = 0$,

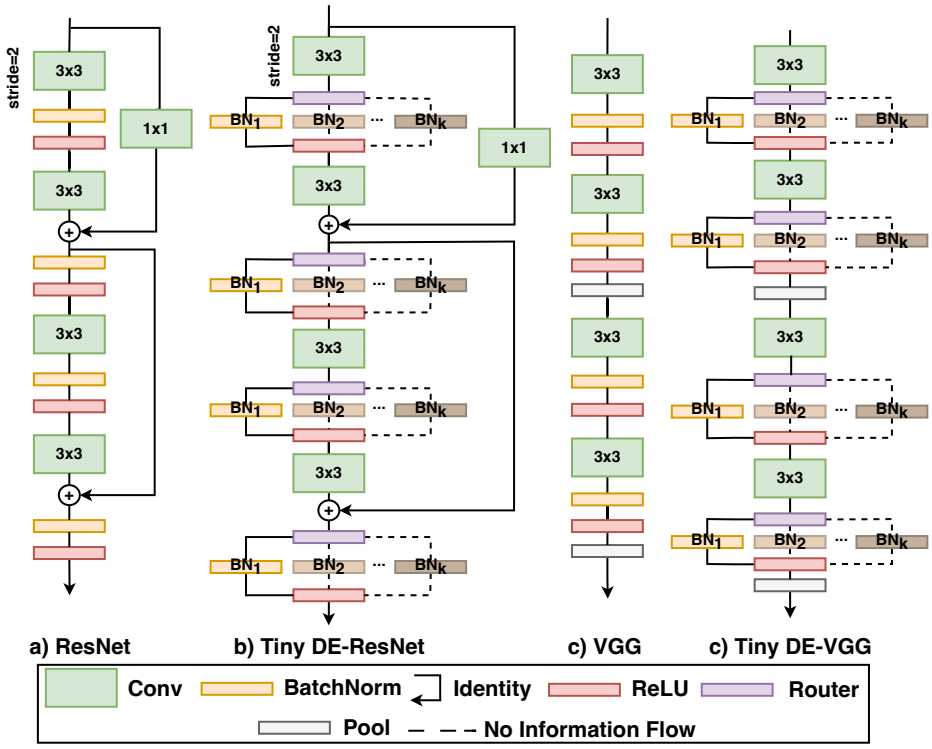


Figure 6.3.: Sketch of proposed Tiny-DE architecture based on popular CNN architectures ResNet [9] and VGG [185]. We only show the four signature layers of a specific topology. Our proposed topology is generalizable across existing topologies, with only the addition of a router before the normalization layers. In the case of our proposed approach in batch mode, no change is required in the topology. **It is recommended to view this figure in color.**

the output $\mathbf{z}^l$ is processed by $\text{BatchNorm}^l_{\gamma_0,\beta_0}$. In the next forward pass, c becomes 1, routing the output $\mathbf{z}^l$ through $\text{BatchNorm}^l_{\gamma_1,\beta_1}$, and this process is repeated until the M-th forward pass. After that, c resets to 0. Note that, due to the global signal routing and synchronization challenge, the counter variable is updated locally in each layer.

This cyclic routing mechanism allows each input of the NN to experience every normalization setting, providing diverse internal-state manipulations within a single inference cycle, which is crucial for enhancing the ensemble's ability to generalize and generate output distribution for uncertainty estimation.

Furthermore, the proposed Tiny-DE can be generalized to all existing NN architectures by making minor modifications, as shown in Fig. 6.3. For popular architectures, such as ResNet and VGG, a router can be inserted after the convolutional layer.

Router Implementation In CiM architectures, the router can be implemented digitally in the periphery using a demultiplexer (DeMux). The DeMux takes the v-bit unsigned counter c as the control signal, allowing for up to 2^v possible routing paths, each corresponding to one of the normalization layers (ensemble members). Since a typical DeMux expects a bit-wise control signal, the DeMux for our purpose is designed to interpret the control signal c in binary representation. This can be expressed as:

$$\text{binary}(c) = b_{v-1}b_{v-2}\cdots b_0, \tag{6.2}$$

where b_{v-1} to b_0 are the bits of the binary sequence representing c.

Our approach requires only changes to the CiM periphery, since the router is implemented in the digital domain with some logic hardware. Specifically, the weight matrix is mapped to the memristor-based crossbar arrays with mapping techniques described in 2.9.1. The MAC operation of a layer is computed in memristor-based crossbar structures (analog domain), and the result is digitized by an analog-to-digital

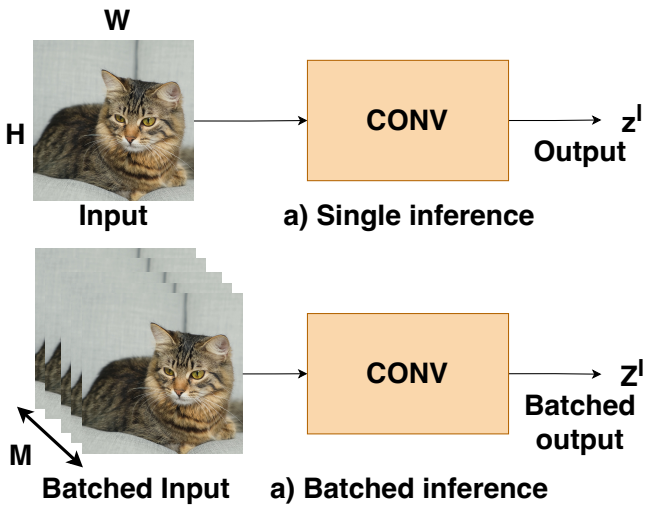


Figure 6.4.: a) Single input processing, and b) batched processing in a convolutional layer. The input is repeated M times to create a batch.

converter (ADC) operation. An accumulator-adder module to sum up the results of all crossbars in a layer (partial matrix multiplication). Following that, the router selects the parameter for normalization, and the normalization is performed. In the following, non-linear activation is performed, and a digital-to-analog (DAC) converts the results of the activation function for MAC operation (in the analog domain) of the subsequent layer. The overall algorithm for our proposed approach in sequential inference mode is depicted in Algorithm 1.

Algorithm 1 Sequential inference mode of Tiny-DE in CiM

Input: Controller c, number of ensembles M, input to the network $\boldsymbol{x}$, number of layers L
for $m = 1, \ldots, M$ **do** ▷ sequential inference
  for $l = 1, \ldots, L$ **do** ▷ single forward pass
    Digital-to-analog conversion
    MAC operation in memristor-based crossbar array
    Analog-to-digital conversion
    Router selects parameters of normalization layer
    Perform normalization
    Non-linear activation
  end for
  Increment counter
end for

6.2.2.2. Single-Shot Uncertainty Estimation

By manipulating the computations for a mini-batch, the computations of the Tiny-DE approach are parallelizable within a hardware architecture that allows batched processing such as CPUs, GPUs, TPUs, and some cases FPGAs [210]. Therefore, only a *single forward pass* with respect to multiple ensemble members in parallel is required to estimate uncertainty. Here, an input to the convolution or linear layer is repeated M times to generate a mini-batch of size M to obtain the batched output $\mathbf{Z}^l$. However, if the batch size of the inference inputs is more than one, e.g., B, by repeating the input similarly M times, an effective batch size of $M \cdot B$ can be created. Therefore, a single forward pass is required for the convolution or linear layer.

However, to still allow a single forward pass through all ensemble members, we propose *EnsembleNorm*.

In EnsembleNorm, the input dimension and the parameters are modified across the batch dimension so that they independently apply normalization to each input of the batch. Specifically, the input of the shape $[M * B, C_{in}, H, W]$ is reshaped as $[M, B, C_{in}, H, W]$. Here, C_{in}, H, and W represent the input channels, height, and width of the input, respectively. Similarly, the learnable parameters expanded to $\beta \in \mathbb{R}^{M \times C_{out}}$ and $\gamma \in \mathbb{R}^{M \times C_{in}}$. That means that the parameters are not only channel-specific, but also unique to each ensemble member. The mean and variance are also calculated in the respective dimensions. That means that each ensemble member $m = 0, \cdots, M-1$ can have its own specific mean $\boldsymbol{\mu}_m$ and variance $\boldsymbol{\sigma}_m^2$. Furthermore, each ensemble member can be scaled and shifted by its own unique parameters, $\boldsymbol{\gamma}_m$ and $\boldsymbol{\beta}_m$.

Subsequently, the normalized output $\bar{\mathbf{Z}}^l$ is reshaped again to $[M * B, C, H, W]$ before applying the non-linear activation function.

Consequently, all ensemble members can compute the output in a single forward pass, eliminating the need to calculate the output of each ensemble member sequentially. Therefore, the computational latency is reduced to a minimum.

6.2.3. Training

The training procedure of Tiny-DE also depends on the operating mode. The sequential mode involves two main phases, but the parallel mode allows single-shot training. Both methods are described in detail in the following.

Sequential Mode As stated earlier, the overall training of the M ensembles requires two main phases. Initially, the full model is trained with all parameters (weights and biases) being updated. After this, the parameters of the model, e.g., weights and biases are frozen, and the normalization layers are re-initialized. Here, "frozen" means that they are not updated using backpropagation. In each subsequent training, only the normalization layers are updated. The training is stopped once a comparable accuracy to the full model is achieved. All trained parameters of the normalization layer are accumulated in a list to allow for ensemble learning as described earlier.

Since the full model is only trained once, the training overhead and complexity are significantly lower compared to [187] and [114], respectively. The decoupling of parameters allows for effective ensemble learning without the overhead of training multiple distinct models from scratch. In addition, it allows one to *obtain M ensemble members from a single pre-trained model.*

Single-Shot Training In the batched processing mode, replacing the normalization layer with the proposed EnsembleNorm layers along with manipulating the dimension as discussed in the earlier section, all the ensemble members can be trained together.

Here, the effective batch size for training may need to be reduced due to the memory overflow issue in GPUs. However, since training is typically done in the cloud, it is not an issue for edge inference.

6.2.4. Prediction and Uncertainty Estimation

The input for inference is forward-passed through the Tiny-DE to get the predictive distribution. The final prediction of Tiny-DE is obtained from the average predictions of all ensemble members.

To obtain uncertainty in the prediction, we explore different methods depending on the task. For classification tasks, the predictive entropy is commonly used, but we also measure the maximum disagreement among the outputs, as shown in Algorithm 2.

Algorithm 2 Maximum Disagreement

```
Input: output samples of y of shape (M, B, K)
Initialize Max Disagreement (MD) with zeros of shape (B, K)
for m = 1, ..., M − 1 do
    for m′ = m + 1, ... M do
        Calculate absolute difference m′ and m output
        Calculate the maximum across the class dimension
        Update Max Disagreement
    end for
end for
```

The Maximum Disagreement metric quantifies uncertainty by calculating the maximum absolute difference in output distributions for each class, across all models in the ensemble. Since it is computed directly from SoftMax output, this metric ranges from 0 to 1. A low maximum disagreement value (closer to 0) indicates low uncertainty, and a high value (closer to 1) indicates high uncertainty.

Furthermore, in semantic segmentation and time series prediction tasks, uncertainty is quantified by the variance in predictions of different ensemble members. Lastly, for regression tasks, the uncertainty is estimated using the negative log-likelihood (NLL) of the prediction.

6.2.5. Diversity Improvement Among Ensemble Members

Diverse predictions among ensemble members are advantageous as they offer complementary perspectives, potentially improving performance and enhancing uncertainty estimates. For our approach, diversity can be improved by a) using different kinds of normalization layers in each member, b) training each ensemble member with different data augmentations, and c) creating multiple bootstrap samples (random samples with replacement) from the training data and training each ensemble member on each sample.

6.3. Results

6.3.1. Experimental Setup

To show scalability on deep learning tasks, we have evaluated our method on four different tasks: image classification, regression, autoregressive time series forecast, and semantic segmentation. To further show scalability on datasets and NN topologies, we have evaluated each task on several state-of-the-art (SOTA) NN topologies (including CNN and RNN) and datasets.

For image classification, we used the CIFAR-10 and CIFAR-100 benchmark datasets on the VGG-19, ResNet-56, ShuffleNet-V2, RepVGG-A1, and TinyML compatible MobileNet-V2 CNN topologies. Furthermore, for the regression task, we have used 10 UCI datasets with a topology and setting as [35]. Specifically, each dataset except for the protein and Year Prediction MSD, is split into 20 train-test folds. Five train-test splits were used for the protein dataset, and a single train-test split was used for the Year Prediction MSD dataset. The NN has 2-hidden layers with ReLU6 nonlinearity followed by a 1D batch normalization layer. The number of neurons is 50 for the smaller datasets and 100 for the larger protein and Year Prediction MSD datasets, making the network compatible with edge AI accelerators. All the dataset was trained for 40 epochs and we have used 5 ensemble members (M=5).

On the other hand, for the time-series forecast, an NN with an LSTM layer and a classifier layer was used for the Mauna Loa CO2 concentrations dataset. Lastly, for Semantic segmentation tasks, we have considered binary as well as multi-class segmentation datasets and two safety-critical scenarios, for biomedical and automotive. For biomedical image segmentation, we have used the Kvasir-SEG [198] dataset, which

Table 6.1.: Results on regression benchmark datasets of the proposed approach and related works Probabilistic back-propagation (PBP) [212], MC-Dropout [35], Deep Ensembles [187] comparing RMSE and NLL. Dataset size (N) and input dimensionality (Q) are also given.

			Avg. Test RMSE and Std. Errors ↓				Avg. Test LL and Std. Errors ↓			
Dataset	N	Q	**PBP**	**MC-Dropout**	**Deep Ensemble**	**Proposed**	**PBP**	**MC-Dropout**	**Deep Ensemble**	**Proposed**
Boston Housing	506	13	3.01 ± 0.18	2.97 ± 0.85	3.28 ± 1.00	**2.97 ±0.46**	2.57 ± 0.09	**2.46 ± 0.25**	**2.41 ± 0.25**	4.92 ±1.03
Concrete Strength	1,030	8	5.67 ± 0.09	5.23 ± 0.53	6.03 ± 0.58	**5.51 ±0.41**	3.16 ± 0.02	**3.04 ± 0.09**	3.06 ± 0.18	5.02 ±0.62
Energy Efficiency	768	8	1.80 ± 0.05	1.66 ± 0.19	2.09 ± 0.29	**1.53 ±0.38**	2.04 ± 0.02	1.99 ± 0.09	**1.38 ± 0.22**	**1.41 ±0.46**
Kin8nm	8,192	8	0.10 ± 0.00	0.10 ± 0.00	0.09 ± 0.00	**0.07 ±0.00**	-0.90 ± 0.01	-0.95 ± 0.03	**-1.20 ± 0.02**	-0.95 ±0.01
Naval Propulsion	11,934	16	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	**0.00 ±0.00**	-3.73 ± 0.01	-3.80 ± 0.05	**-5.63 ± 0.05**	-3.81 ±0.08
Power Plant	9,568	4	4.12 ± 0.03	**4.02 ± 0.18**	4.11 ± 0.17	4.48 ±0.18	2.84 ± 0.01	2.80 ± 0.05	**2.79 ± 0.04**	2.95 ±0.05
Protein Structure	45,730	9	4.73 ± 0.01	4.36 ± 0.04	4.71 ± 0.06	**3.92 ±0.03**	2.97 ± 0.00	2.89 ± 0.01	**2.83 ± 0.02**	5.05 ±0.52
Wine Quality Red	1,599	11	0.64 ± 0.01	0.62 ± 0.04	0.64 ± 0.04	**0.64 ±0.05**	0.97 ± 0.01	**0.93 ± 0.06**	0.94 ± 0.12	1.28 ±0.33
Yacht Hydrodynamics	308	6	**1.02 ± 0.05**	1.11 ± 0.38	1.58 ± 0.48	3.22 ±1.59	1.63 ± 0.02	1.55 ± 0.12	**1.18 ± 0.21**	1.37 ±0.43
Year Prediction MSD	515,345	90	8.88 ± NA	8.85 ± NA	8.89 ± NA	**8.53 ±NA**	3.60 ± NA	3.59 ± NA	**3.35 ± NA**	7.63 ± NA

contains medically obtained gastrointestinal polyps images on the Feature Pyramid Network (FPN) [166]. For automotive scene understanding we used the CamVid [199] dataset, which consists of road scene images and involves segmenting each pixel into one of the 12 classes on the UNet++ topology [211]. We have further evaluated the generalized scene understanding task with the Pascal VOC dataset with the fully convolutional network (FCN). The encoder network for each topology is shown in brackets in Table. 6.3.

Note that the semantic segmentation task is known to be more challenging than other tasks due to its finer granularity. That is, it involves segmenting an image into multiple sections and assigning each pixel with its corresponding class label.

The performance of the classification task is evaluated on inference accuracy, time series, and regression on root-mean-square-error (RMSE), and semantic segmentation on pixel accuracy and mean intersection-over-union (mIoU) metrics.

In terms of uncertainty estimation, classification tasks are evaluated on data distribution shift and out-of-domain data as OOD data. Specifically, for data distribution shift, images are corrupted by 90° rotation and Gaussian noise, a subset of the CIFAR-C dataset [20]. Furthermore, SVHN (Street View House Numbers) and STL-10 datasets are used for out-of-domain data, which refers to data that significantly deviate from the distribution of the training data. The predictive entropy distribution is calculated from the mean of 250 batch samples and is subsequently modeled as a normal distribution.

6.3.2. Evaluation of Regression on Real-World UCI Datasets

The result of the regression task is depicted in Table 6.1. Our approach is compared with Bayesian [212], implicit ensemble (MC-Dropout) [35], and ensemble [187] methods. As can be seen, our method outperforms or is competitive with existing methods in terms of RMSE and NLL. Specifically, our method outperforms other methods in 8 out of the 10 datasets in terms of RMSE. In some datasets, we observe that our method is slightly worse in terms of NLL. We believe that this is due to the fact that our method optimizes for RMSE instead of NLL (which captures predictive uncertainty). We found that there is a trade-off between RMSE and NLL. Optimizing for NLL instead reduces RMSE. Also, we did not perform hyperparameters optimization, unlike [35] which performed grid search.

6.3.3. Evaluation of Classification

In classification tasks with various topologies, it can be observed that our method improves inference accuracy by up to 0.81% or is comparable with the single model, as shown in Table 6.2.

In terms of uncertainty estimates in the OOD data, Fig. 6.5 shows the predictive uncertainty of the ResNet-32 model trained on clean CIFAR-10. It can be observed that the predictive entropy is low in clean CIFAR-10,

Table 6.2.: Performance of Tiny-DE with CIFAR-10 and CIFAR-100 dataset trained on various topologies with up to 15 ensemble members.

Topology	Dataset	Number of ensembles			
		1	5	10	15
VGG-19	CIFAR-10	93.91	93.86	93.79	93.80
ResNet-56		94.37	94.28	94.14	94.38
ShuffleNet-V2		93.3	93.27	93.44	93.67
RepVGG-A1		94.93	94.56	94.84	94.62
MobileNet-V2		94.05	93.67	93.92	94.01
VGG-19	CIFAR-100	73.87	74.21	74.56	74.68
ResNet-56		72.63	72.64	72.85	72.82
ShuffleNet-V2		72.58	72.75	73.54	73.11
RepVGG-A1		76.44	75.77	74.67	75.21
MobileNet-V2		74.29	74.41	74.67	75.21

that is, ID data. However, if the model receives OOD data, e.g., rotated, SVHN, or STL-10 data, the predictive entropy increases from baseline. Importantly, the relative change in the predictive entropy is significantly higher for our proposed Tiny-DE approach. Here, the relative change in the uncertainty estimates signifies better capabilities in the uncertainty estimates. Furthermore, the change in predictive entropy becomes greater as the number of ensembles increases, which is an ideal behavior.

In contrast, the CIFAR-100 model is evaluated on the max disagreement metric, as shown in Fig. 6.6. In ID data, our approach shows finer granularity in uncertainty estimates. Specifically, the uncertainty is low for correctly classified images and high for incorrectly classified images. On corrupted (rotated and noisy) images, OOD data, the model can still predict some images correctly. Our approach shows a similar uncertainty distribution for correctly and incorrectly predicted images. In addition, the relative change from the baseline distribution is also high. In domain-changed data (SVHN and STL-10), our approach shows high uncertainty with distributions concentrated toward the right. Furthermore, the distributions shift more toward the right as the number of ensembles increases.

6.3.4. Evaluation of Time-Series Prediction

The performance of our proposed approach on autoregressive time series prediction is shown in Fig. 6.7. As can be seen, the prediction curve is closer to the ground truth for our approach compared to the single model. Furthermore, the curve approaches ground truth as the number of ensemble members increases. Specifically, the single model achieves an RMSE score of 0.1119. In contrast, our proposed Tiny-DE method achieves an RMSE score of 0.0943 for 5 ensemble members, which is reduced to 0.0921 for 10 members. That translates into a 17.7% reduction in the RMSE score. In general, all models follow the same trend as the ground truth.

6.3.5. Evaluation of Semantic Segmentation

Similarly, in semantic segmentation tasks with several challenging datasets and SOTA models, our approach performs comparably or outperforms the baseline model, as shown in Table 6.3. Two qualitative examples of each dataset are shown in Fig. 6.8. As can be seen, the predictions are close to the ground truth, with only incorrect predictions around the edges of segments or in uncommon classes. Here, uncommon classes refer to classes that occur infrequently or are less represented in the dataset.

In terms of uncertainty estimates, our proposed approach can estimate uncertainty accurately. In an ideal case, misclassified pixels should have high uncertainty around them, and correctly classified pixels should have low uncertainty. As shown in Fig. 6.8 our approach captured this behavior effectively.

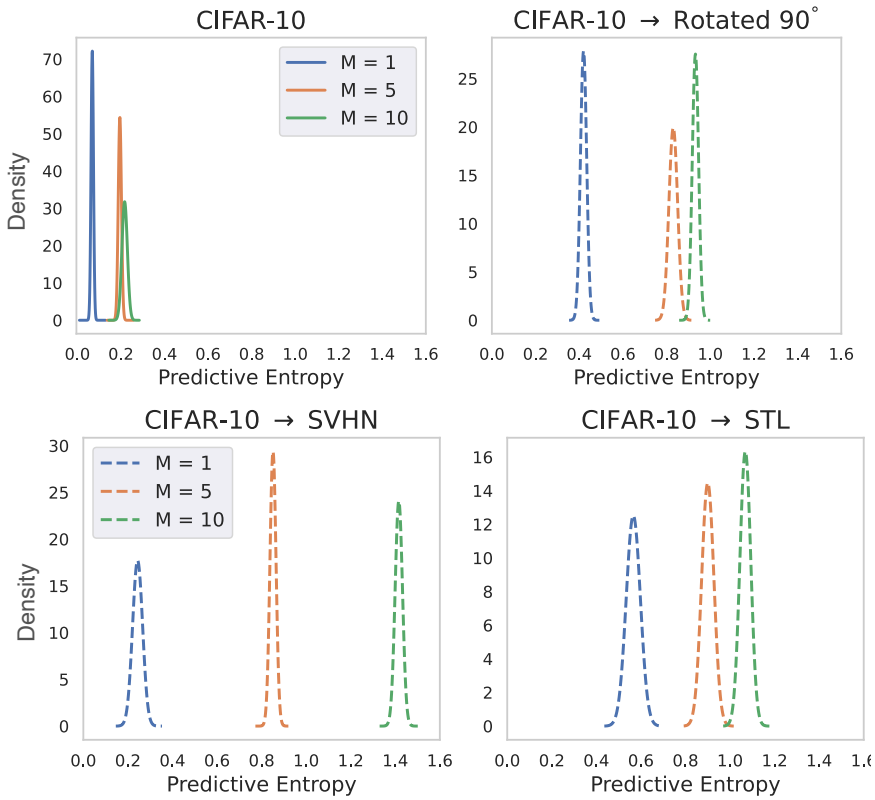


Figure 6.5.: Uncertainty distributions for the Tiny-DE approach on CIFAR-10, including ID CIFAR-10, and OOD datasets such as rotated CIFAR-10, SVHN, and STL. Notably, larger ensembles show increased relative change of uncertainty distribution from ID compared to a single model (M = 1). **It is recommended to view this figure in color.**

Table 6.3.: Pixel accuracy and mean intersection over union (IoU) of the single model and our proposed Tiny-DE (M = 5) with different datasets and SOTA models.

Topology	Dataset	Single Model		Proposed (M=5)	
		Pixel Acc	mIoU	Pixel Acc	mIoU
UNet++ (ResNet-34)	CamVid	91.65	63.95	91.52	63.99
FPN (ResNet-18)	KvaSir	95.95	74.62	95.89	74.57
FCN (ResNet-50)	CIFAR-10	87.78	69.63	87.71	68.58

6.3.6. Comparison with Related Works

In the presence of OOD data, the higher the relative change in predictive entropy with respect to ID distribution, the better the method. Compared to related uncertainty estimation methods with model ensemble [187, 35, 114], the relative predictive entropy of our Tiny-DE approach is much higher, as shown in Fig. 6.9. This further underscores the robustness of our approach. Here, the validation is done on the ResNet-32 topology on the CIFAR-10 dataset, but we found that this translates to other topologies and datasets.

6.3.7. Improving Diversity

As mentioned in Section 6.2.5, more diversity among the prediction of the ensembling members can lead to better performance and uncertainty estimates. Therefore, we performed another set of experiments in which each ensemble member was trained on different data augmentations. We found that by improving diversity with different random data augmentations to train each ensemble member, the uncertainty estimates increase in OOD data. For example, as shown in Fig. 6.10, the uncertainty maps around incorrect

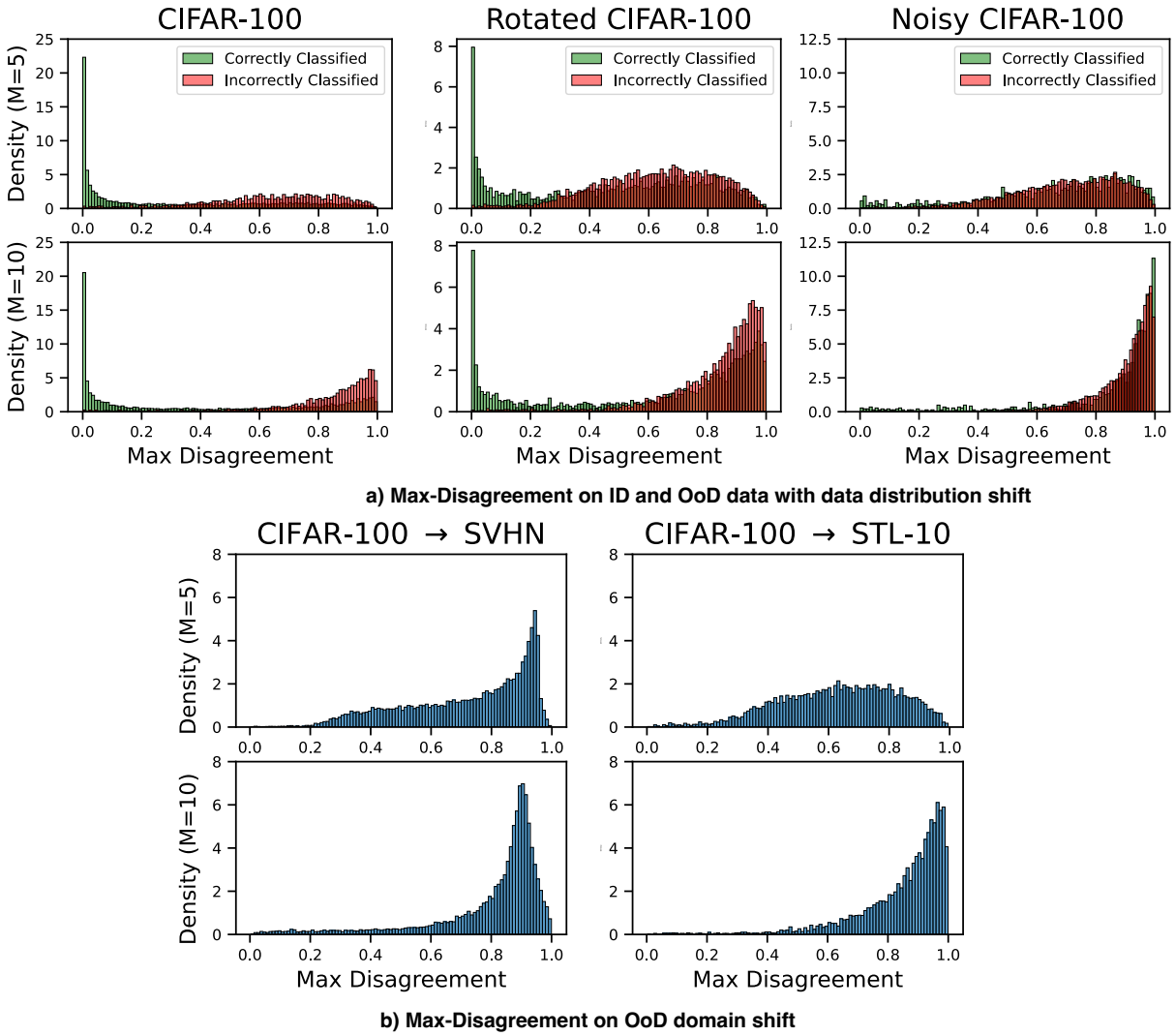


Figure 6.6.: ID and OOD Max Disagreement distributions for the Tiny-DE approach trained on clean CIFAR-100 (ID). Notably, larger ensembles show increased relative change of uncertainty distribution from ID. **It is recommended to view this figure in color.**

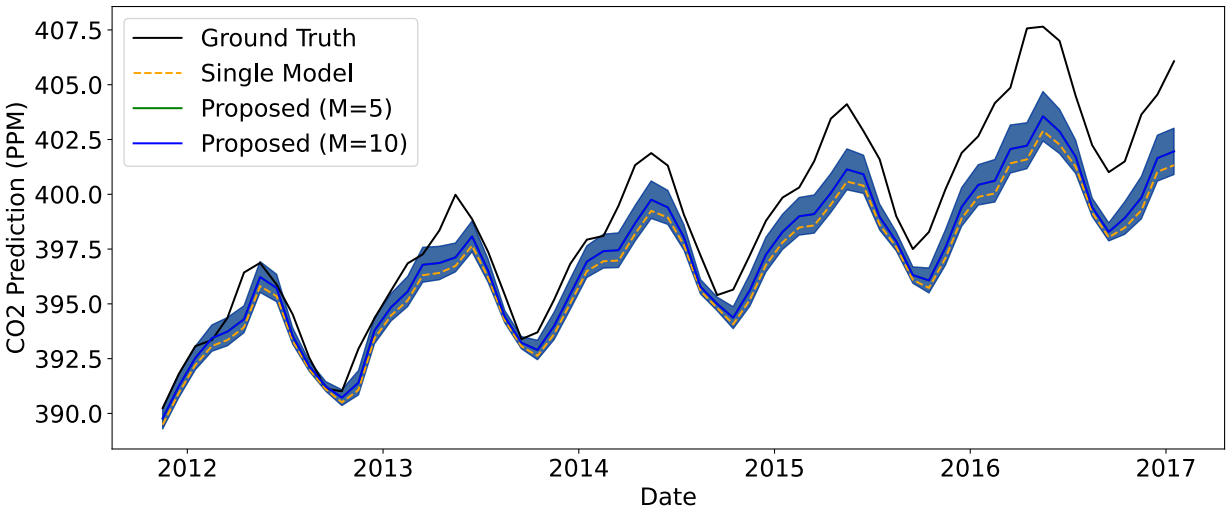


Figure 6.7.: Auto-regressive time series prediction of atmospheric CO2 of a single model and our proposed Tiny-DE model with up to 10 ensemble members. The shaded region shows the uncertainty around prediction. **It is recommended to view this figure in color.**

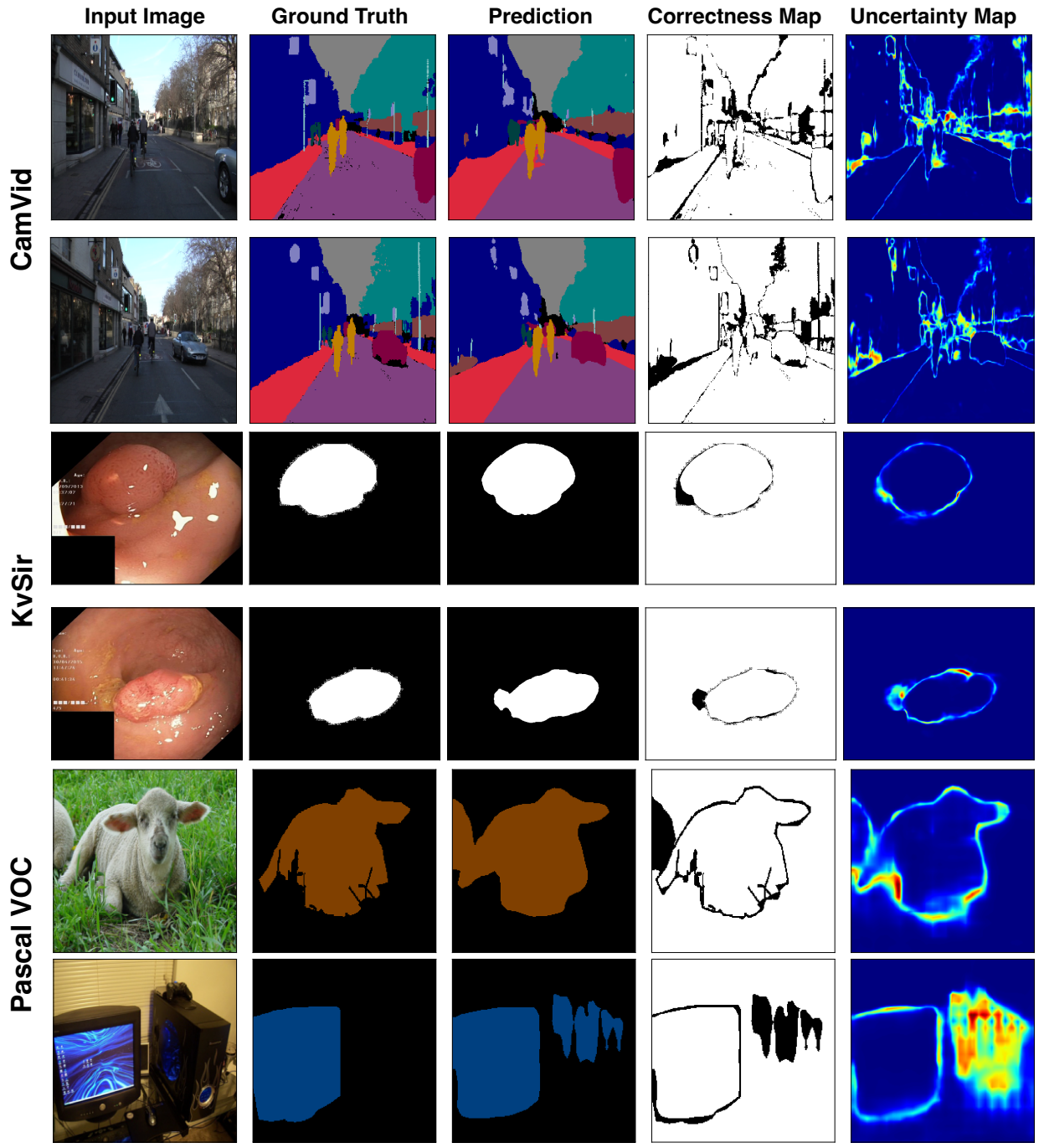


Figure 6.8.: Qualitative results for several semantic segmentation tasks and associated uncertainty estimates. The correctness map is a binary diagram indicating correct and incorrect predictions in white and black, respectively. **It is recommended to view this figure in color.**

pixels become stronger compared to Fig. 6.8 when each ensemble member is trained using different data augmentations. Furthermore, pixel accuracy and mIoU increased to 88.67% and 72.48%, respectively.

6.3.8. Hardware Overhead

Figs. 6.11 shows the relative cost in terms of memory and latency of our approach and related approaches for the ResNet-32 topology. In terms of memory overhead, our approach has approximately the same overhead as the BatchEnsemble [114] and MC-Dropout [35] methods but significantly outperforms branch ensemble [113] and Deep Ensemble [187] methods. The memory overhead of the deep ensemble increases

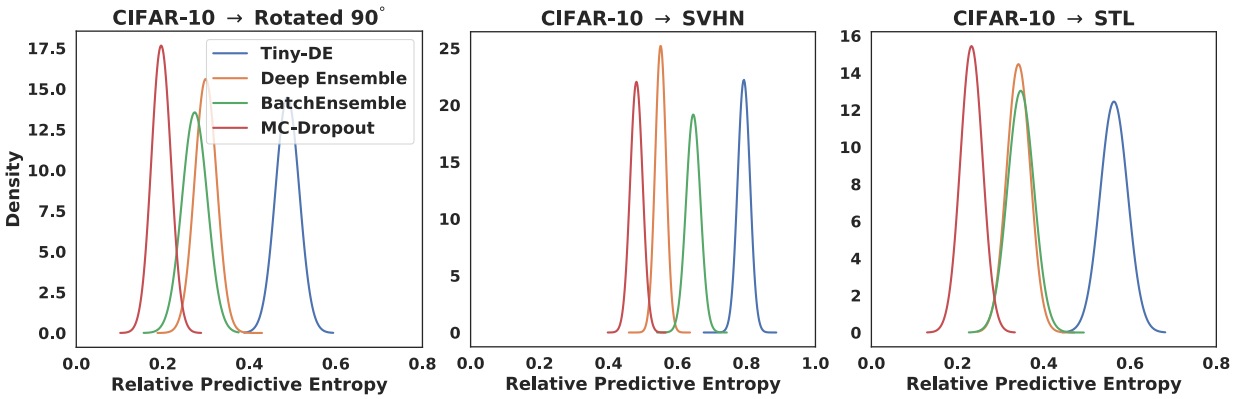


Figure 6.9.: Relative change in predictive entropy on OOD data of Tiny-DE (ours) in comparison to Deep Ensemble [187], MC-Dropout [35], and BatchEnsemble [114]. **It is recommended to view this figure in color.**

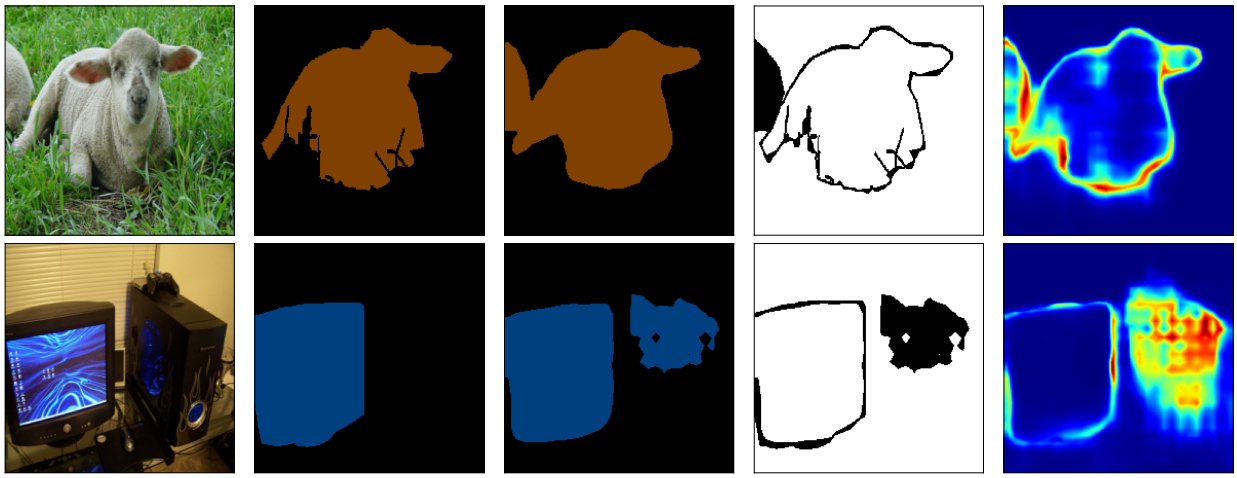


Figure 6.10.: Results for Pascal VOC with improved diversity in ensemble members using different random data augmentation. **It is recommended to view this figure in color.**

linearly with the size of the ensemble. In the branch ensemble method, the last two convolutional and final classifier layers are ensembled. Since the last two layers consume ~ 75% of the total parameters, ensembling them leads to a high memory overhead. Specifically, if batch normalization is used, our method has slightly more overhead compared to BatchEnsemble due to the requirements of running mean and

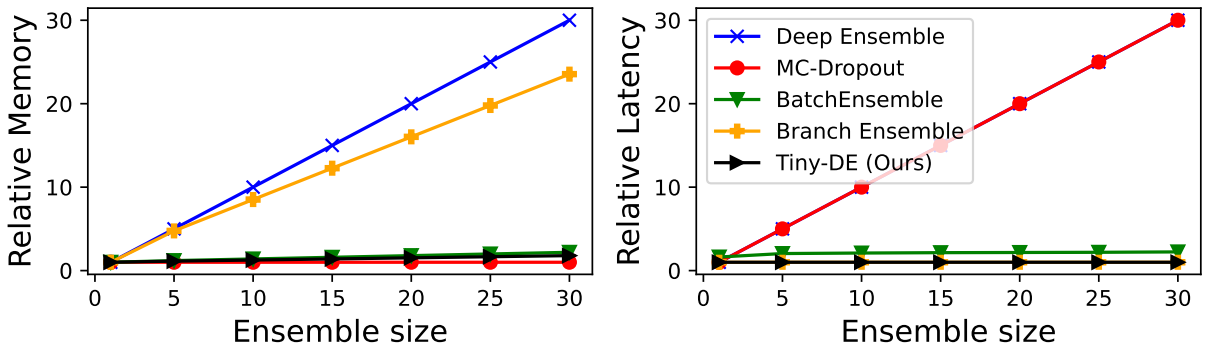


Figure 6.11.: The inference cost in terms of memory and latency of our and related approaches w.r.t the ensemble size. The results are relative to a single model cost. The testing time cost and memory cost of the naive ensemble are plotted in blue. **It is recommended to view this figure in color.**

variance vector storage. However, for other normalization layers that do not calculate running mean and variance, the memory overhead is the same.

In terms of latency, our approach has the same latency as the single model, as no additional computation is required relative to the single model. Therefore, the latency is the same as the branch ensemble method. However, BatchEnsemble has additional computation requirements in the input and output of convolutional layers, leading to as much as 2× latency as our method. The latency of the deep ensemble and the MC-dropout increases linearly with the size of the ensembles.

In general, our approach provides a good balance between memory and latency. In parallel mode, our approach requires *one forward-passes and has approximately the same memory overhead* relative to a single model (*an ideal case*). Consequently, our approach has up to $\sim M\times$ reduction in overhead.

6.4. Scientific Impact of This Work

We outline the scientific impact of this work and the main contributions can be summarized as follows:

- **Ensembling normalization layer:** For resource constraint edge AI accelerators, ensembling where it does not hurts, that is normalization layers, while sharing other parameters is an attractive approach for low-cost uncertainty estimation.
- **Vectorized Processing for Single-Shot Uncertainty Estimation** In edge AI accelerator architecture that allows batch processing, e.g., GPUs, TPUs, and so on, vectorizing input can lead to single-shot uncertainty estimation. To this end, this work proposed a method with the EnsembleNorm. EnsembleNorm layer allows normalizing all ensemble members in a single shot.
- **Scalability** The proposed Tiny-DE network topology is scalable to existing NN topologies, AI accelerator architectures, NN tasks, and datasets. For a scalable solution, considering edge AI accelerator architecture is an effective approach.
- **Potential for Real-World Applications** Our evaluation shows our approach lead to substantial reduction in computational and storage requirements without sacrificing accuracy and quality of uncertainty estimates. Therefore, it is an attractive option for real-world safety-critical applications, where real-time decision-making under uncertainty is paramount.

6.4.1. Section Conclusion

In this section, we present a cost-effective ensembling method for edge AI accelerators. We introduce the Tiny-DE topology, where only normalization layers are ensembles, and all ensemble members share the weights and biases. Our approach is scalable in terms of AI accelerators, datasets, NN topologies, and tasks. With an expensive evaluation, we show that our approach can estimate uncertainty effectively with up to $\sim$ 1% improvement in accuracy, and a 17.7% reduction in RMSE score on various tasks. Furthermore, our approach has up to $\sim M\times$ reduction in hardware overhead.

Part II.

Methods for Quantifying Functional Uncertainty of Edge AI Accelerators

While uncertainty estimation in prediction is important in increasing confidence in predictions and detecting OOD, it is also crucial to detect faults and variations in computational elements, memory storing weight, and activations for functional safety. This is because edge AI accelerators are prone to non-idealities such as manufacturing and in-field defects and variations, as well as environmental factors such as temperature fluctuations. Such factors introduce uncertainty in the parameters, activations, and overall predictions of edge AI accelerators, which can significantly impact their accuracy and reliability.

In safety-critical applications, where incorrect predictions can have catastrophic consequences, it is imperative to rigorously test and validate the functionality of edge AI accelerators. Therefore, proper in-field functional testing is required to ensure functional safety and reliability. Functional testing can quantify the uncertainty of core operations in Edge AI Accelerators.

This thesis targets testing for hardware non-idealities in edge AI accelerators using explicit and concurrent testing methods. Explicit testing methods are designed to identify faults and variations during the development phase using the "pause-and-test" method. Where the system is non-functional while testing. Therefore, reducing testing costs, specifically testing latency via test vector compression, is crucial as many AI applications do not tolerate long system downtime. We made several contributions in this regard to reduce the testing overhead to the absolute minimum possible. Furthermore, we target testing edge AI accelerators implementing stochastic BayNN. Testing stochastic BayNNs is challenging due to their stochastic output for the same input. We aim to address this problem.

We also target concurrent testing methods, which can test the edge AI system concurrently without system downtime. Concurrent testing enables runtime monitoring and fault detection during the operation of the AI accelerator, ensuring continuous and reliable operation. However, it is challenging to test an edge AI accelerator concurrently while still achieving a high true positive rate (fault coverage) and low false positive rates. This thesis proposed a concurrent testing method to address this.

Lastly, disentangling the source of uncertainty is important to perform root cause analysis and apply a targeted uncertainty mitigation strategy. For example, uncertainty due to OOD data requires reviewing the input by an expert annotator, but uncertainty due to hardware non-idealities requires either re-training, re-calibration, or hardware replacement. Therefore, this also targets to address this issue.

This part presents all our solutions to the above-mentioned challenges.

7. Explicit Testing of NNs

This chapter presents the explicit testing methods considered in this thesis. To reiterate, the main aim is to reduce the number of test vectors stored on the chip to reduce the storage overheated. In addition, our aim is to reduce test queries that reduce overall testing overhead, such as the number of MAC operations, testing latency, and power consumption.

7.1. Approximate Gradient Ranking

In this section, we consider using training images to test edge AI accelerators and compress test vectors. However, achieving high coverage with low test queries and finding test vectors that are most sensitive to parameter changes is a challenge. In addition, it is significantly challenging to test "hard-to-detect" faults. We offer several solutions to overcome the challenges. The section is based on our IEEE ITC paper [41].

7.1.1. Methodology

In this section, we first provide details and the overall flow for generating functional test vectors with the proposed approximate gradient ranking (Method ❶). Following this, we present a further approximated method (Method ❷) for the functional test generation targeting pre-trained NN models.

7.1.1.1. Approximate Gradient Ranking

During NN training, the mini-batch stochastic gradient descent algorithm is usually used for learning due to its benefits, such as easier fitting of the training data to memory, computational efficiency, vectorization, and so on. A gradient δ of a parameter $\mathbf{W}$ is calculated with respect to loss $\mathcal{L}$ for a mini-batch $\mathcal{D}_B(\subset \mathcal{D}_{train})$ of size B in each step, where, $\mathcal{D}_{train}$ is the total training dataset. However, some training inputs are harder to predict than others and will require more tuning of the NN parameters than others. We hypothesize that the training input that requires more tuning of the NN parameters than others throughout the training will be more sensitive to changes in the NN parameters than others. This property can be utilized to identify training data that can be used as functional test vectors.

Our goal is to generate functional test vectors that will result in a large change in NN predictions with a small perturbation or faults in NN parameters when the testing dataset is applied. Therefore, it will be easier to detect subtle faults in the memristive-mapped NN. In this paper, we use our hypothesis to rank each of the training data to generate functional test vectors (Method ❶) to test the memristor-based NN. The overall algorithm for the training steps is shown in Algorithm 3. To implement the proposed gradient ranking, we first label each of the training data $1 \cdots Q$ so that their respective gradient accumulation can be tracked (line 2). To calculate gradient ranking, we need to calculate and accumulate the sum of absolute gradients $\sum|\delta|$ of all of the NN parameters θ for all the data points of $\mathcal{D}_{train}$ throughout the training. However, such calculations are computationally inefficient and will require a significantly larger memory size for training. The inefficiency comes from the creation of B gradient matrices for each of the NN parameters for a mini-batch of size B instead of one, as done normally in a single training step.

Although back-propagation can be done for each training data point serially to build one gradient matrix per NN parameters, it will lead to a longer training time as B back-propagations will be required. Also, the benefits of mini-batch stochastic gradient descent mentioned earlier are not maintained.

Therefore, we approximate the required gradient calculation with the change in loss $\mathcal{L}$ value in each training step for each data point of $\mathcal{D}_{train}$ (lines 9-11). The suggested change in $\mathcal{L}$ value in each training step for a single training example can be described as

$$\Delta\mathcal{L} = |\mathcal{L} - \tilde{\mathcal{L}}|. \tag{7.1}$$

Here, $\tilde{\mathcal{L}}$ is the loss after each optimization step (parameter update) (line 10). This adds only a few steps to normal NN training (lines 9-11) and is much more efficient in finding the desired functional test data using our proposed approach. The computation graphs that are usually created for back-propagation are not required.

The intuition behind our proposed approximation is that $\Delta\mathcal{L}$ is proportional to δ, since δ is calculated with respect to loss $\mathcal{L}$. Therefore, a higher δ value will lead to a higher $\Delta\mathcal{L}$ value. In turn, the higher the $\Delta\mathcal{L}$ for training data, the higher the NN parameter tuned for those data.

Algorithm 3 Proposed Functional Test Generation Algorithm

1: **Require:** initial NN parameters θ_0, loss function $\mathcal{L}$, mini-batch size B, training dataset $\mathcal{D}_{train} \subset (X, Y)$, learning rate η.
2: **Results** test dataset $\mathcal{D}_{test}$, trained parameters θ
3: label training dataset $X^n, n = 1 \cdots N$
4: Initialize $\delta\mathcal{L}$ of each training data to zero
5: **for** epoch **in** epochs **do**
6: Sample minibatch $\mathcal{D}_k (\subset \mathcal{D}_{train})$ of size B
7: Forward pass $\mathcal{D}_k$ on $\mathcal{F}$ with parameter θ
8: Calculate elementwise loss $\boldsymbol{\mathcal{L}}$ for each x_B, $B = (1 \cdots B)$
9: Backpropagate and update parameters θ based on average loss
10: Forward pass on $\mathcal{D}_B$ on $\mathcal{F}$ with updated parameter $\hat{\theta}$
11: Re-calculate elemntwise loss $\tilde{\boldsymbol{\mathcal{L}}}$ on updated parameters $\hat{\theta}$
12: Calculate and accumulate element wise change in loss $\Delta\mathcal{L}$
13: **end for**
14: Sort the training data $argsort(\mathcal{D}_{train})$ based on $\Delta\mathcal{L}$ to get ranking
15: Sample $S (\subset \mathcal{D}_{train})$ data point with maximum $\Delta\mathcal{L}$ values

We accumulate $\Delta\mathcal{L}$ throughout the training (line 12). After training, we rank each training data point based on the accumulated $\Delta\mathcal{L}$ value (line 14). We sample a small subset of the training data as the test data $\mathcal{D}_{test} \subset \mathcal{D}_{train}$ (line 15). Please note that for our objective, the gradient δ of each NN parameter for each input is needed, as opposed to δ of input with respect to $\mathcal{L}$ that are usually required for adversarial examples generation. As shown in Figure 7.1, the top-ranked NN prediction does not change at all (completely overlaps) when an input with the lowest $\Delta\mathcal{L}$ is used for testing the NN, while the NN prediction changes from label '6' to label '2' by using an input with the highest $\Delta\mathcal{L}$. This shows similar behavior to the method proposed in [37, 117] with synthetic test data.

Our proposed functional test generation procedure treats the NN as a black box and is non-invasive. The NN training procedure is similar to typical DNN training, except that a few additional simple steps are added that do not hinder normal NN training and can be considered as a validation step on the training dataset. The training curve for the NN training with and without the proposed additional step is shown in Figure 7.2. The training curve for the proposed method is the same as that for normal training.

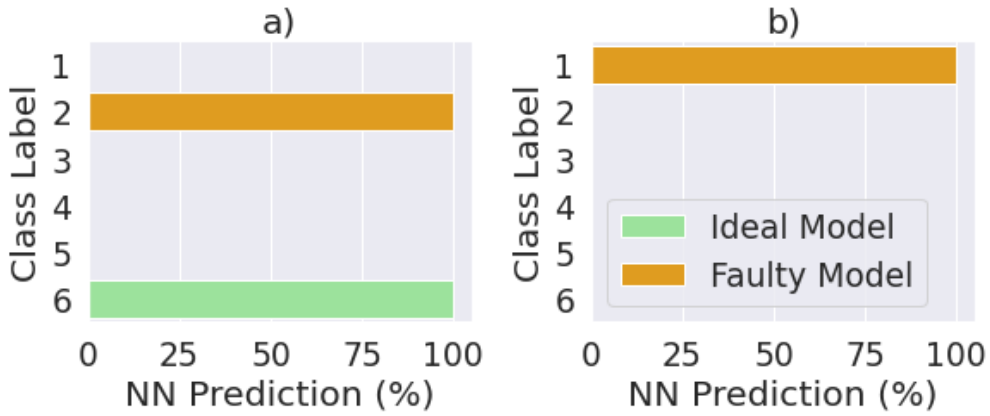


Figure 7.1.: The change in NN prediction probability of NN on a faulty and ideal model (fault-free) when the input is (a) test input (images with highest ($\Delta\mathcal{L}$)), (b) normal input (images with lowest ($\Delta\mathcal{L}$)). NN prediction probability changes significantly on test inputs compared to normal input. **It is recommended to view this figure in color.**

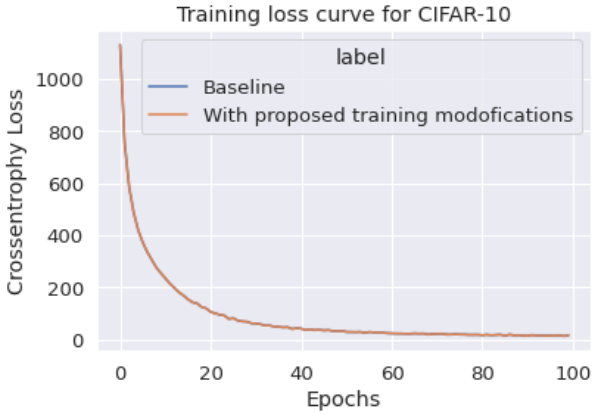


Figure 7.2.: The training curve of baseline (typical DNN training) and proposed approximate gradient ranking method. **It is recommended to view this figure in color.**

7.1.1.2. Regularization with Data Augmentation

To reiterate, data augmentation can help NN training to achieve better generalization. Although our proposed method works quite well without data augmentation, we propose to use data augmentation during training and functional test generation, as we show later that using data augmentation makes test data more sensitive to changes in NN parameters. However, our proposed method does not rely on data augmentation, since it is generally not used in MLP training and tasks such as regression.

7.1.1.3. Global Approximation

Although the generation of the functional test patterns using the proposed approximate gradient ranking method is simple and effective (shown later in Section 7.1.4) it can not generate test patterns for a pre-trained model. We, therefore, propose to apply further approximation to our proposed gradient ranking method (Method ❶) discussed earlier. Instead of tracking the change in gradient for each input precisely throughout the training, it can be approximated with a single step. The change in gradient between a NN with randomly generated parameters and a pre-trained NN can also be utilized to rank training data points that can be used as functional test vectors. We define this method as the global approximate method (Method ❷) for test generation.

Generating test vectors with the global approximation method is similar to Method ❶ and Algorithm 3. The back-propagation (line 8), parameter update (line 8), and the accumulation of change in loss (line 11)

are not required as the NN is already trained. The $\Delta\mathcal{L}$ is calculated in a single step and can be described as:

$$\Delta\mathcal{L} = |\mathcal{L}_{\mathrm{I}} - \tilde{\mathcal{L}}_{\mathrm{I}}|. \tag{7.2}$$

Here, $\mathcal{L}_{\mathrm{I}}$ and $\tilde{\mathcal{L}}_{\mathrm{I}}$ are the loss of randomly initialized and pre-trained NN, respectively. Therefore, this method still considers the NN as a black-box, non-invasive, and model-agnostic. No knowledge about the pre-trained model is required for test vector generation. The losses $\mathcal{L}_{\mathrm{I}}$ and $\tilde{\mathcal{L}}_{\mathrm{I}}$ can be easily calculated by downloading a pre-trained model and resetting its parameters.

However, this method has its own limitations. Specifically, it is limited to larger NNs and normalized models, i.e., normalization layers and data normalization are used. Data normalization makes sure that the input features have a zero mean and a unit variance. These methods reduce the sensitivity of random weight initialization and typically lead to similar performance. In addition, this approach requires access to the training data of the pre-trained model.

7.1.2. Proposed Test Application Method

In this section, we provide details about the proposed test application method for fault detection. We show how the proposed approach works to detect two categories of faults: important faults (Method ①) as well as hard-to-detect faults (Method ②). Following this, we present a test time label generation method to achieve 100% accuracy on the test dataset without overfitting them. Furthermore, the overall flow for the fault detection framework that can detect important and hard-to-detect faults is also presented.

7.1.2.1. Categorizing Fault Induced Accuracy deviations

Due to the non-idealities of memristors, the inference accuracy of the implemented NN can change to a varying degree of severity. Observable faults that cause noticeable degradation in inference accuracy are more important to detect as they violate acceptable accuracy margin and hence should be detected as a part of manufacturing testing. We define such faults as important faults. If such faults are undetected, and they can accumulate and become catastrophic. Catastrophic faults can cause drastic degradation in the accuracy of deep learning applications. For periodic in-system testing, we want to detect faults early enough before they become catastrophic. Therefore, our fault detection method primarily depends on the detection of faults that cause task-related accuracy to drop below the accuracy margin ΔA_{infer} due to defects, variations, and non-idealities of memristors.

Conversely, in certain safety-critical applications such as autonomous driving and medical image analysis, even a small accuracy fluctuation, e.g., $\Delta A_{infer} = 0.1\%$, is considered catastrophic and unacceptable. This type of fault is defined as the hard-to-detect fault. However, detecting such small changes in NN is more expensive and is hard in comparison, analogous to small delay defects in digital circuits [213]. We, therefore, propose a specialized method that can detect even a tiny change in NN accuracy.

7.1.2.2. Detection method for important faults

In this work, the accuracy of the test dataset is used as the output response analysis for fault detection. That is, we determine the change in the accuracy of the test dataset to detect memristive faults (Method ①). A fault is detected if the accuracy of the test dataset deviates from the ideal fault-free accuracy. However, the accuracy of the test dataset is not guaranteed to achieve 100%, especially when regularization is used during training which does not allow overfitting of the training data. We, therefore, propose to substitute miss-predicted test vectors with the NN predicted label for a classification task to bring the accuracy of the test data set to 100% without overfitting them as done in [37]. For manufacturing testing, the labels should

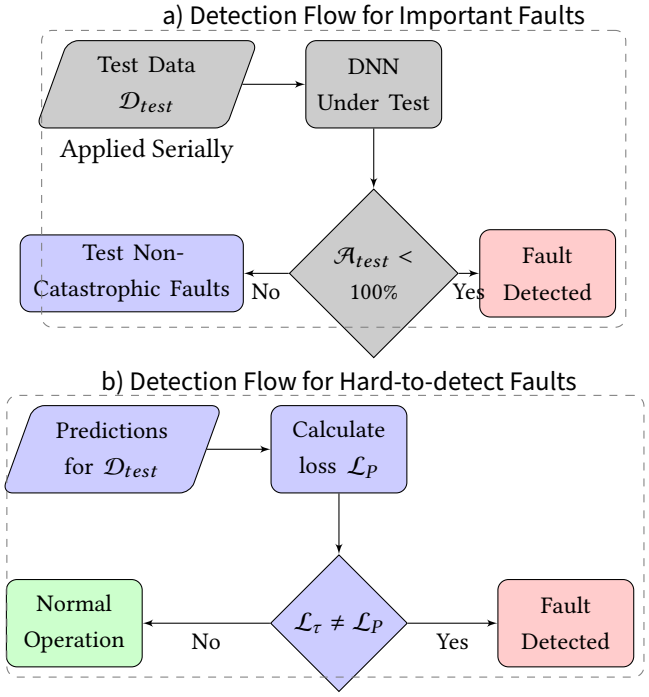


Figure 7.3.: Fault detection flow for a) Important faults and b) Hard-to-detect faults. Test quarries are applied only once for both fault detection methods. Detection of hard-to-detect faults is carried out conditionally after an important fault detection step.

be generated pre-mapping the NN on the memristive accelerator, i.e., on the trained model. For periodic in-system testing applications, labels should be generated post-mapping after manufacturing defects are mitigated.

Finding the optimal size of the testing dataset is challenging since that depends on many factors, including fault severity, fault type, and NN task, as discussed in Section 7.1.4. During fault detection, normal device operation is paused. Hence, it is of paramount importance to reduce the testing time. In CiM architecture, typically only one test input can be applied to the memristive accelerator at a time. We refer to each test input as test queries. Since our goal is to determine accuracy deviation on the test dataset, a fault is detected when even one test query is miss-predicted, and the testing procedure of the memristive NN stops similar to digital IC testing. This can significantly reduce the required test queries and testing time. Furthermore, this solves the challenge of finding the optimal size of the test dataset. The upper end of the allowed test data can be stored but the number of test queries required will be automatically adjusted based on factors such as fault severity, fault type, and NN task.

7.1.2.3. Detection method for hard-to-detect faults

The proposed fault detection method can achieve high test coverage if the faults cause noticeable degradation in inference accuracy, e.g., $\Delta A_{infer} = 2\%$. However, when the accuracy degradation is tiny, e.g., $\Delta A_{infer} = 0.1\%$, the proposed important fault detection method fails to detect all of the faults (as shown later in Section 7.1.4), leading to fault escapes that are not tolerated in safety-critical applications. Also, if the number of test vectors stored in the hardware is small, e.g., $16 - 32$, there are more test escapes for the proposed important fault detection method (Method ①).

We, therefore, propose to store the average loss $\hat{\mathcal{L}}_\tau$ of the testing dataset and compare the loss $\mathcal{L}_P$ of the faulty NN to detect faults (Method ②). Hence, it requires a single value storage in addition to storing test data, which adds negligible storage overhead. Please note that this method does not require full access to device outputs but only the predicted score, e.g., the confidence score of the predicted label in a classification task. Therefore, this method can not be applied to a device that only outputs the predicted label of a classification task, but just one stage before that output.

Table 7.1.: Shows the inference accuracy and data augmentation setting of different datasets used in this work.

Dataset	Topology	Data Augmentation	Accuracy
Fashion-MNIST	MLP	No	89.36%
CIFAR-10	ResNet-18	Yes	92.94%
	ResNet-18	No	86.08%
CIFAR-100	ResNet-110	No	73.75%

7.1.3. Fault Modelling and Injection Framework

7.1.3.1. Modelling Conductance Variations

The conductance variations depend on the type of memristor device and environmental factors such as temperature. In this paper, the variation model proposed in [75] is used. Variation in the memristor device is therefore modelled into two types: multiplicative and additive Gaussian and can be described as:

$$G_{real} = G_{ldeal} + \mathcal{V}(G_{real}, X), X \sim \mathcal{N}(\mu, \sigma^2) \tag{7.3}$$

Where, G_{real} and G_{ideal} are the deviated and ideal conductance of the memristive cells, respectively. Also, the function $\mathcal{V}(.)$ models the variations and $\mathcal{N}$ is the probability density function representing normal distribution. The weights of the pre-trained model are encoded as the conductance of the memristor cells G_{ideal}. Therefore, the G_{ideal} is considered as the weights of pre-trained model W_{orig} and G_{real} as the variation injected weight W_{noisy} and can be described as

$$W_{noisy} = W_{orig} + \mathcal{V}(W_{orig}, X), X \sim \mathcal{N}(\mu, \sigma^2). \tag{7.4}$$

The conductance fluctuations due to additive and multiplicative type variations are further modeled as two factors: device-to-device manufacturing variations (spatial fluctuation) $\mathcal{N}_S \sim (1, \sigma_S^2)$ and thermal variations (temporal fluctuation) $\mathcal{N}_T \sim (\eta_0, \sigma_T^2)$ [74]. Where, η_0 is the noise scale that is used to control the severity of noise. The overall fault model for multiplicative and additive variations can be defined according to [128] as:

$$W_{noisy}^{MUL} = W_{orig} + W_{orig} \cdot \mathcal{N}_T \cdot \mathcal{N}_S \tag{7.5}$$

$$W_{noisy}^{ADD} = W_{orig} + \mathcal{N}_T \cdot \mathcal{N}_S \tag{7.6}$$

Based on Equations 7.6 and 7.5 the variations are injected into the weight matrix of all layers as random noise. For each fault run, we randomly sample from the Gaussian distribution of $\mathcal{N}_T$ and $\mathcal{N}_S$.

7.1.3.2. Modelling Permanent Faults

Permanent faults such as Stuck-on/off, Stuck-open/short, Slow-write, and Read/Write disturbance can be modelled as:

$$W_{stuck} = f(\mathcal{P}_{stuck}, W_{orig}) \tag{7.7}$$

where, $\mathcal{P}_{stuck}$ is the percentage of permanent faults injected and function $f(\cdot)$ is the fault model. In this paper, we model $f(\cdot)$ by randomly sampling $\mathcal{P}_{stuck}$% bits of weights and randomly setting them to either 1 or 0. However, for Read/write disturbance, we flip random $\mathcal{P}_{stuck}$% bits of weights.

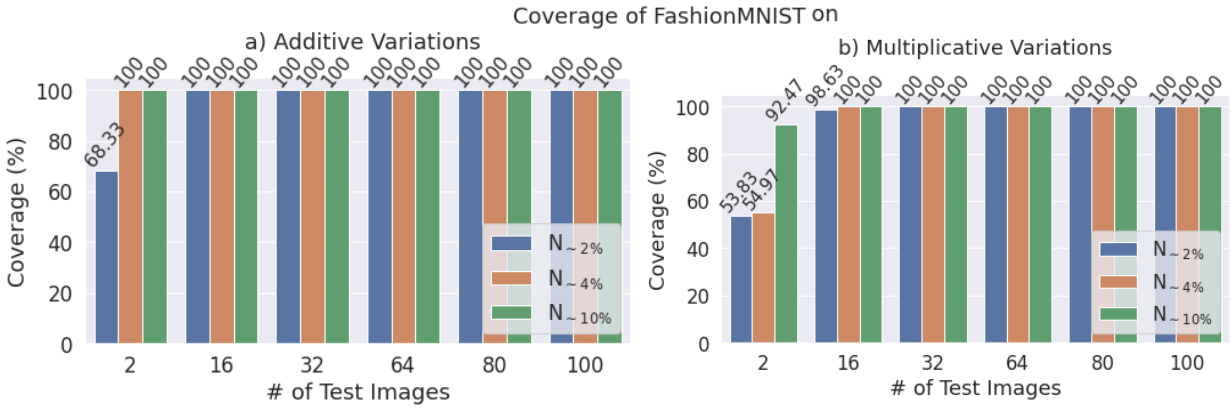


Figure 7.4.: Test coverage of Fashion-MNIST dataset considering a) additive, and b) multiplicative variations. **It is recommended to view this figure in color.**

7.1.4. Results

7.1.4.1. Evaluation Setup

We have trained an MLP with four layers (256 neurons per hidden layer) on the Fashion-MNIST and ResNet-18 CNN topology on the CIFAR-10 benchmark datasets. We have used a cross-entropy loss function for the classification task and optimized the NNs with the ADAM optimization algorithm with the default setting in Pytorch. The best accuracy achieved on the validation dataset is used for simulation. For each dataset, we have trained three models (M_1, M_2, M_3) from scratch with a different random seed. We used an already trained ResNet-110 model on the CIFAR-100 dataset to evaluate the global approximation method. Specifically, the model parameters are downloaded from [214] without knowledge of training.

We have not applied any pre-processing to the Fashion-MNIST and CIFAR-100 datasets. However, we have augmented the CIFAR-10 training dataset with RandomHorizontalFlip and RandomResizedCrop type data augmentation methods due to their benefits mentioned earlier. Furthermore, as shown in Table 7.1, applying data augmentation improves the inference accuracy of the CIFAR-10 dataset by $\sim 7\%$. We also discuss the effect of data augmentation on testing coverage. Augmentation is not applied to the inference datasets.

In this paper, we quantize the NN to 8-bit precision that represents 256 quantization levels and encoded them to eight memristor cells. The trained weights of the linear layers are mapped to a crossbar with a dimension (m and n) of 256×512, and the fully unrolled weight matrixes of the convolution layers are split across multiple different crossbars of the same dimension as the linear layers. We have used the weight matrix splitting and current accumulation as proposed in [64].

In this paper, we define the accuracy margin of important fault as $\Delta A_{infer} = 2\%$, i.e., the fault that causes inference degradation of more than 2%. We refer to the catastrophic accuracy degradation as $\Delta A_{infer} > 5\%$.

Without losing generality across different memristor technology, we have simulated our method on 1000 memristive crossbars instances to simulate the effect of per-chip variations and permanent defects. The same amount of simulations are performed for both CNN and MLP with all the benchmark datasets, models (M_1, M_2, M_3), and test queries. The strength of conductance variation is increased by changing the η_0 value of Equations 7.6 and 7.5 for multiplication and addition type variations to analyze the impact of different inference accuracy deviation scenarios. We have normalized the noise strength by choosing η_0 such that inference accuracy degrades to $\sim$ 2%, $\sim$ 4%, and $\sim$ 10%. We refer to those cases $N_{\sim 2\%}$, $N_{\sim 4\%}$, and $N_{\sim 10\%}$, respectively, in this paper. Furthermore, for the read/write disturbance and stuck-at/slow write faults, we randomly inject $\mathcal{P}_{stuck} = 0.2\%$ faults based on Equation 7.7 defined in Section 7.1.3.2. We

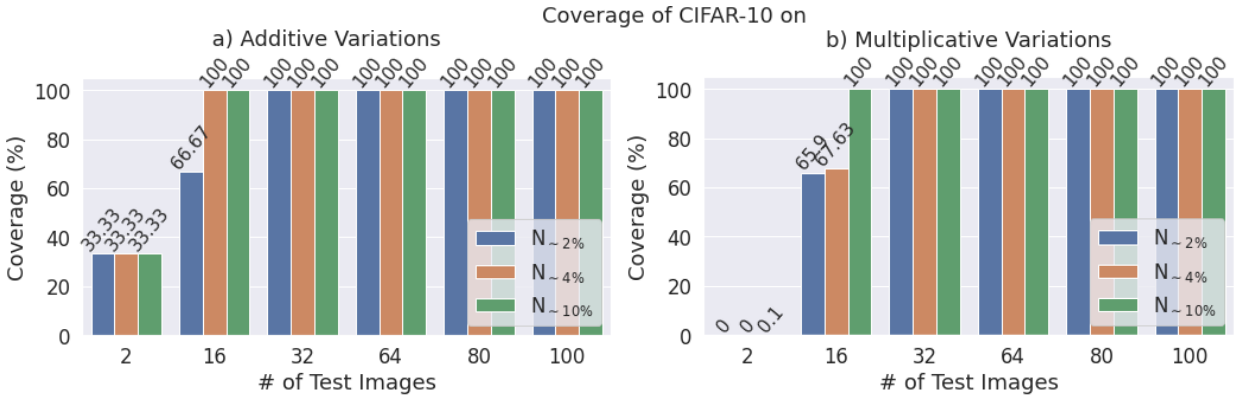


Figure 7.5.: Test coverage of CIFAR-10 dataset considering a) additive, b) multiplicative variations. **It is recommended to view this figure in color.**

report test coverage as the ratio between detected faults ($A_{test} < 100\%$) and overall faults ($A^N_{infer} < A_{infer}$, $N \subset N_{\sim 2\%}, N_{\sim 4\%}, N_{\sim 10\%}$) and can be described as

$$coverage = \frac{\#of(A_{test} < 100\%)}{\#of(A^N_{infer} < A_{infer})} \times 100. \tag{7.8}$$

Here, A^N_{infer} refers to the inference accuracy of a faulty NN and A_{infer} is the inference accuracy of a fault-free NN. In this paper, we refer to the sensitivity of the testing dataset as a qualitative measure of the ability to detect parameter deviations of memristive deep learning applications.

7.1.4.2. Analysis of Test Performance of Important Fault Detection

Analysis of Variation Detection When a sufficiently large number of functional test data is stored, that is, $\mathcal{D}_{test} \geq 16$, the proposed method achieves 100% test coverage (see Equation 7.8) in both the Fashion-MNIST and CIFAR-10 datasets under both multiplicative and additive variation, as shown in Figures 7.4 and 7.5. Also, we have found that the testing dataset is more sensitive to additive variations compared to multiplicative variations in both datasets. For example, the coverage of Fashion-MNIST with only two stored test vectors can reach 100% when the accuracy degradation is sufficiently higher (>= 4%). Furthermore, the test vectors are also more sensitive to MLPs on the Fashion-MNIST dataset compared to CNN with CIFAR-10 datasets. Specifically, the test coverage of CIFAR-10 with multiplicative variations is 0%, but the Fashion-MNIST dataset achieves up to 92.47% coverage when only two test vectors are stored on the hardware.

The number of test queries required is generally small, e.g., only 4 maximum number of queries are required for the Fashion-MNIST dataset (as shown in the Figure 7.7 (a)) to achieve 100% coverage, therefore, the coverage reaches 100% at 16 test vectors stored. Similarly, the CIFAR-10 dataset requires a maximum of 31 test queries (as shown in the Figure 7.7 (b)) to achieve 100% coverage and with only 19 test vectors, coverage can reach 99.97%. CNNs require more test queries compared to MLPs because it has considerably more parameters and the topology is deeper.

Permanent Faults The proposed test pattern generation method can achieve a test coverage of 100% under permanent faults when the stored test vectors are sufficiently large, i.e., $\mathcal{D}_{test} \geq 64$ for the CIFAR-10 dataset as shown in Figure 7.6 (a). However, MLP with Fashion-MNIST dataset requires only 16 test vectors stored to achieve 100% coverage as shown in the Figure 7.6 (b).

The number of test queries required is again generally small, but they are relatively higher compared to the case when testing conductance variations, e.g., only 27 maximum number of queries required (as shown

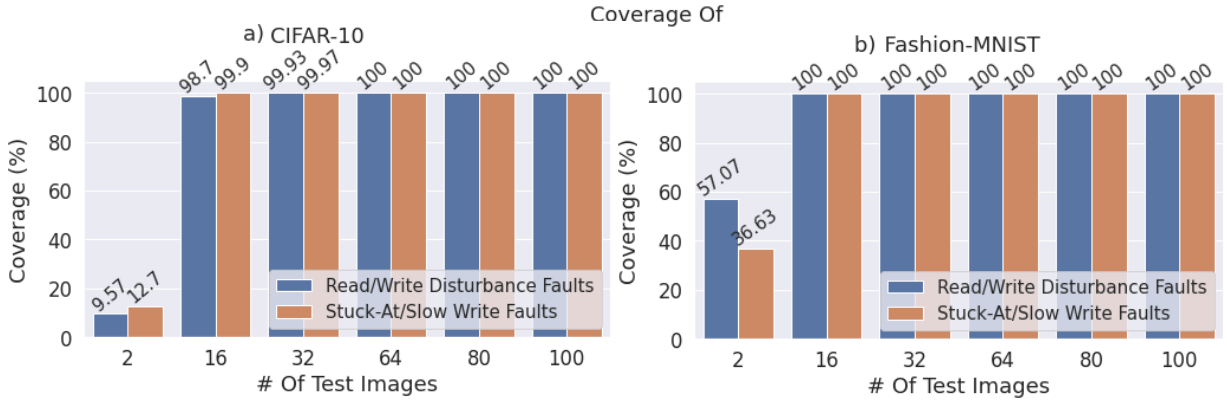


Figure 7.6.: Test coverage of permanent faults with a) CIFAR-10, and b) Fashion-MNIST dataset considering both read/write disturbance and stuck-at/slow write faults. **It is recommended to view this figure in color.**

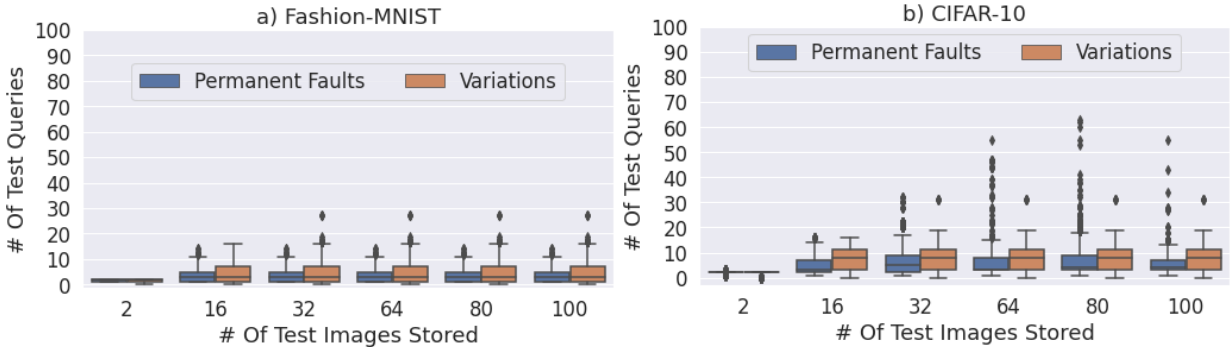


Figure 7.7.: Number of test queries required in relation to the number of test vectors stored for a) Fashion-MNIST dataset, b) CIFAR-10 dataset. Analyses were performed on all the permanent faults and both variations types. **It is recommended to view this figure in color.**

in Figure 7.7 a)) for the Fashion-MNIST dataset to achieve 100% coverage. As a result, the coverage of Fashion-MNIST reaches 100% when 32 test vectors are stored. In addition, with only 15 test queries, 99.99% test coverage can be achieved. Similarly, the CIFAR-10 dataset requires a maximum of 63 test queries (as shown in Figure 7.7 b)) and with only 18 test queries the coverage can reach 99.04%. The proposed test vectors are again more sensitive when testing MLPs with the Fashion-MNIST dataset compared to CNNs with CIFAR-10 datasets.

7.1.4.3. Analysis of Test Performance of Hard-To-Detect Faults

Our proposed important fault detection method with 64 generated test vectors can achieve 100% even when accuracy is degraded due to faults is 0.27% for both CNN and MLP with CIFAR-10 and Fashion-MNIST datasets as shown in the Figure 7.8. However, when the accuracy degradation is significantly lower. For example, when the average accuracy degradation due to multiplicative variations is only 0.08%, the proposed important fault detection method (Method ①) only achieves 65% coverage, but the proposed hard-to-detect method (Method ②) can achieve 100% coverage for the CIFAR-10 dataset. Furthermore, the proposed hard-to-detect method produces results similar to the Fashion-MNIST dataset.

The maximum number of test queries required increases to detect important faults when the accuracy degradation due to the deviation of the NN parameter is low ($< 0.5\%$). For example, 55 test queries are required when the inference accuracy degradation of the Fashion-MNIST dataset is 0.21% compared to 5 for an accuracy degradation of 25.6%. At higher accuracy degradation, only one maximum test query is required to achieve 100% coverage. However, the full number of stored test vectors is used as test queries for hard-to-detect faults.

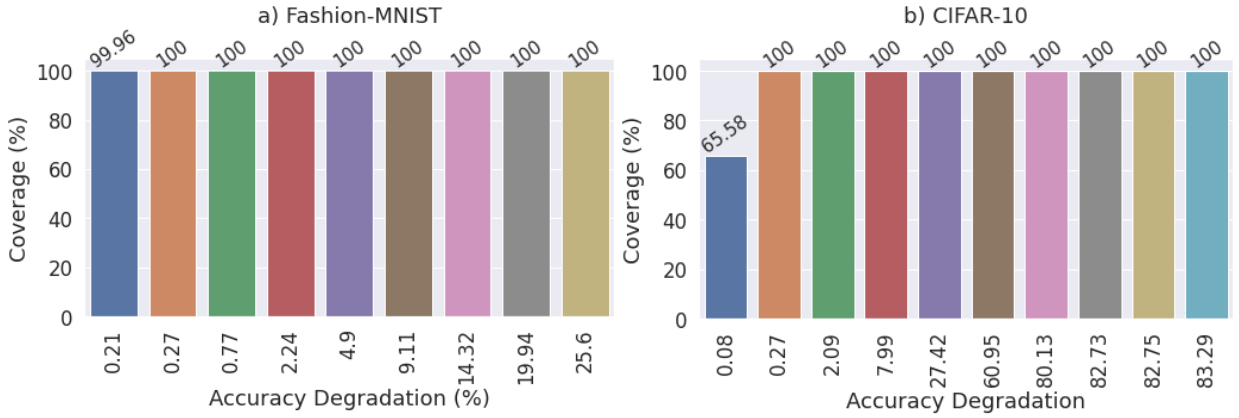


Figure 7.8.: Analysis of test coverage when the accuracy degradation of a) fashion-MNIST dataset, b) CIFAR-10 dataset is varied up to 0.27%. Here, the size of the test vector stored is 64, and multiplicative type fault injection is performed.

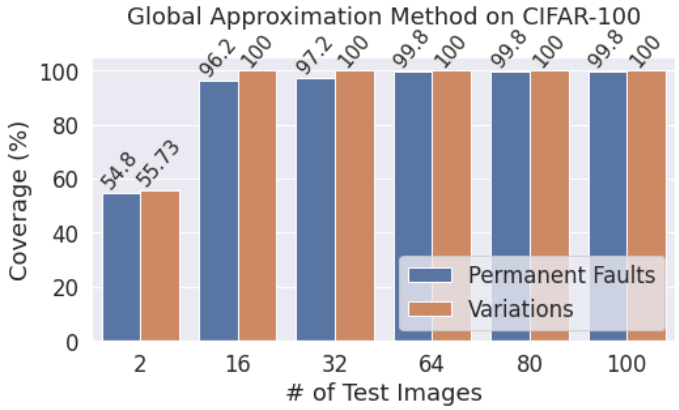


Figure 7.9.: Analysis of test coverage of global approximation method on pre-trained CIFAR-100 dataset. **It is recommended to view this figure in color.**

When random data augmentation is not applied to the CIFAR-10 training, the sensitivity of the test data decreases. For example, the coverage of permanent faults decreases to 91.4%, and the lower end of detectable accuracy due to faults increases to ~ 4%. Therefore, we recommend augmenting the data during training.

7.1.4.4. Analysis of the Global Approximate Ranking

Our proposed global approximation method (Method ❷) to generate test vectors can still achieve a test coverage of 100%, as shown in Figure 7.9 for memristive defects and variation on the pre-trained CIFAR-100 dataset. Similarly, the coverage results are comparable to the test vector generated with a more precise approximate gradient ranking (Method ❶) in the CIFAR-10 dataset.

7.1.4.5. Comparison with State of the Art and Overhead Analysis

We have evaluated our work with related work [37], [117], and [118] that uses the functional test generation method. The analysis is done based on the numbers reported in their paper. We have used the same bit-width to store the images and test labels as in [37]. There is a negligible difference between the size of test vectors and the storage overhead of our work and work in [118], but our method achieved up to 16% more test coverage. On top of that, our method has significantly lower storage overhead and test vector size compared to [37, 117] but still manages to outperform them and achieve 100% test coverage, as summarized in Table 7.2. We have normalized the number of test queries required to achieve test coverage similar to [37].

Table 7.2.: Comparison of the proposed method with the related work in terms of test coverage, memory storage overhead (when re-training data is and is not available), number of test queries required, and fault-detection resolution. The analysis is done on CNN with the CIFAR-10 dataset.

	[37]	[117]	[118]	Proposed
Size of test vectors	10000	1024	**10-50**	16-64
# of test queries (normalized)	10000	1024	10-50	**17**
Memory overhead (MB)	0.015[1]			**0.000256**[1]
	245.775[2]	234.42[2]	0.154[2]	**0.1966**[2]
Fault detection resolution (%)		~3		**~ 0.27**
				~ 0.01
Coverage (%)	99.27[3]	98[3]	76[3]/ 84[4]	**100**[4]

[1] Re-training data is stored in hardware.
[2] Re-training data is not stored in hardware.
[3] Synthetic testing data
[4] Original training data used as testing data

7.1.5. Scientific Impact of This Work and Contributions

We outline the scientific impact of this work as follows:

- **Compact Test Pattern Generation:** The work presented in this section showed that test pattern compaction and non-invasive testing method is an attractive research direction to reduce overall testing and can benefit pre-trained models that are treated as intellectual property. Thus, the testability of resource-constrained edge AI accelerators can be improved.
- **Secure Testing:** The method proposed in this technique does not require full access to NN output but only final prediction or image label, which increases security against adversarial attacks [215].
- **Scalability of Testing Methods:** To improve the applicability of a testing method, it is important that the testing method is scalable across various NN topologies, datasets, and non-idealities types. Our evaluation showed that it is a viable option in this research direction.
- **Practicality:** A practical test generation method is one that is easy to implement and requires minor modifications to the training and inference of the NN. The proposed approach requires only one additional step during training that does not alter the original training process. Thus, it is an attractive option in this regard.

7.1.6. Section Conclusion

In this section, we propose a comprehensive test generation flow that ranks and selects a small subset of input data from the training dataset for the functional test with significantly lower overhead compared to previous solutions [37, 117, 118] to test the memristive NN implementations. Furthermore, we also propose test application methods based on the severity of faults. Our method requires only a few additional steps to generate the functional test pattern and can achieve up-to 100% test coverage with only 0.128% of the training data. Consequently, it reduces the complexity of the test patterns generation and has a negligible memory requirement for testing regardless of the types of faults. Our work allows for easier and faster detection of faults before they negatively affect the accuracy of deep learning applications.

7.2. Single-shot Testing Large-Scale Deep Neural Networks

While we were able to significantly reduce the number of test vectors and test queries compared to related works, in this section, we aim further to reduce the testing cost to the absolute minimum. We aim to test the whole NN model in an AI accelerator using one test vector and one forward pass, which is the extreme end of test compression. However, there are challenges in generating such a test vector and defining an online testing method for this purpose. In this section, we propose solutions to overcome the challenges of single-shot testing.
The section is based on our journal paper from IEEE TCAD [42]

7.2.1. Methodology

7.2.1.1. Motivation and hypothesis of our approach

To reiterate, in the presence of faults or variations in the parameters of memristor-mapped NNs, their representation changes from the expected (trained) parameters, resulting in degradation of their performance. According to Equation 2.1, non-ideal parameters will directly affect the weighted sum and, in turn, the activation of a layer. Since activation of a layer becomes the input to the following layer, the cascading effect of non-ideal parameters is likely to ultimately flow to the overall output $\hat{y}$ of the NNs. Therefore, the distribution logits $\mathbf{z}^L$ are also expected to change, as shown in Fig. 7.10(c).

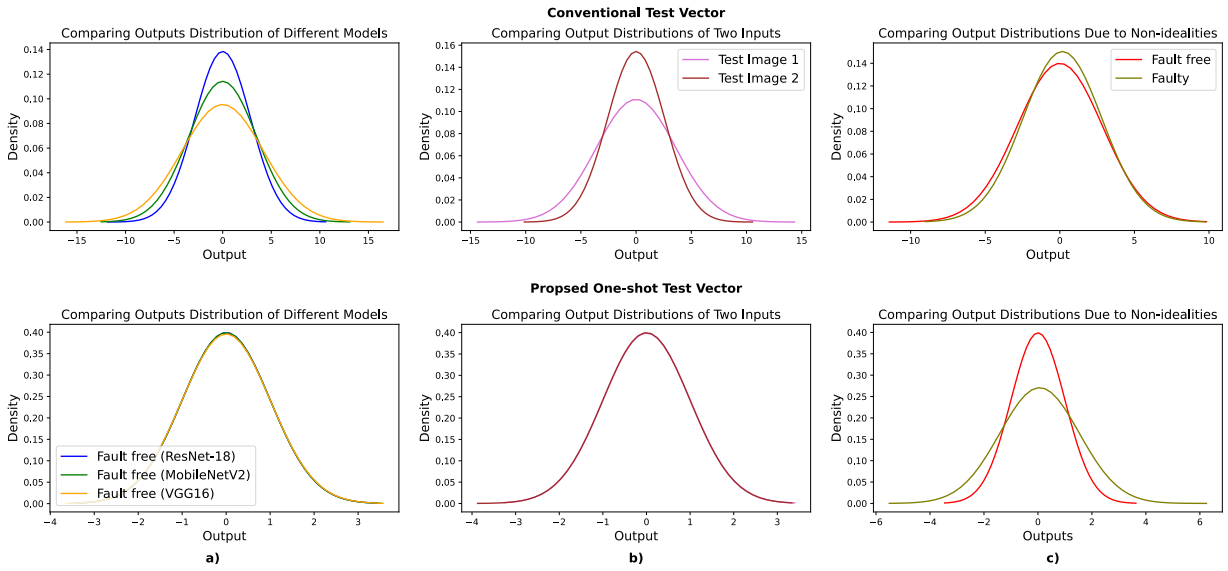


Figure 7.10.: a) Change in output distribution depicted for different NN models but on the same test vector, b) comparison of the change in the output distribution for two different test vectors but on the same NN model (ResNet-18). While the conventional method reveals a change in output distribution across different models and test vectors, our approach ensures standardized output distributions (distributions overlap) irrespective of models or test vectors. c) We compared the relative change in output distribution for the same noise level between the proposed and conventional test vectors. The output distribution is more sensitive to noise for our proposed one-shot test vector. In the conventional method, the test vectors are randomly sampled from the ImageNet validation dataset. **It is recommended to view this figure in color.**

Therefore, we introduce a novel *one-shot testing* method based on the observation and hypothesis that faults and variations in the memristive NN parameters influence the distribution of $\mathbf{z}^L$. Our approach aims to detect distribution shifts in the logits of the model using a single test vector that is specifically designed to produce a distinct output distribution for fault-free cases. Consequently, faults and variations can be easily detected by evaluating the output distribution of a memristive NN after applying the one-shot test vector.

However, there are several challenges associated with this approach. The primary challenges include standardizing the output distribution using one test vector, estimating the change in distribution for pre-trained models, and designing an effective one-shot test vector for various model architectures. We discuss them in the following section with their respective solutions.

7.2.1.2. Proposed deviation detection method

Since the expected distribution of a model is unknown and likely varies from one model to another and from one test vector to another (as shown in the upper half of Fig. 7.10(a) and (b)), it is difficult to estimate the change in distribution for a pre-trained model. Therefore, we propose standardizing the logits distribution of each *model under test* (MUT) to a unit Gaussian distribution, $\mathbf{z}^L \sim \mathcal{N}(\mu \approx 0,\ \sigma^2 \approx 1)$, which means zero mean $\mu \approx 0$ and unit variance $\sigma^2 \approx 1$. Standardizing the output distribution is crucial for the one-shot testing method, as it ensures a consistent and comparable metric across various models and test vectors. Also, it reduces the likelihood of false-positive deviation detection and enhances the sensitivity to non-ideal parameters.

Let $\mathbf{z}^L \sim \hat{\mathcal{N}}(\mu,\ \sigma^2)$ be the output distribution of a memristive NN model. Here, output distribution refers to the row distribution of an NN (logits distribution) before applying any final activation function, such as SoftMax or Sigmoid. Faults and variations in the parameters of memristive NNs can be detected by evaluating the Kullback–Leibler (KL) divergence between the expected output distribution $\mathcal{N}$ and the output distribution of memristive NNs as:

$$D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) = \sum_{i=1}^{n} \hat{\mathcal{N}}(i) \log \frac{\hat{\mathcal{N}}(i)}{\mathcal{N}(i)} \tag{7.9}$$

which can be simplified for two normal distributions as:

$$D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) = \log \frac{\sigma_{\mathcal{N}}}{\sigma_{\hat{\mathcal{N}}}} + \frac{\sigma_{\hat{\mathcal{N}}}^2 + (\mu_{\hat{\mathcal{N}}} - \mu_{\mathcal{N}})^2}{2\sigma_{\mathcal{N}}^2} - \frac{1}{2}. \tag{7.10}$$

We assumed that the distributions $\hat{\mathcal{N}}$ and $\mathcal{N}$ were discrete, since we quantized the parameters of the memristive NN. Here, we denote the mean and standard deviation of the output distribution $\hat{\mathcal{N}}$ of the memristive NN as $\mu_{\hat{\mathcal{N}}}$ and $\sigma_{\hat{\mathcal{N}}}$, respectively. Similarly, $\mu_{\mathcal{N}}$ and $\sigma_{\mathcal{N}}$ represent the mean and standard deviation of the expected output distribution $\mathcal{N}$. Since $\mu_{\mathcal{N}}$ and $\sigma_{\mathcal{N}}$ are defined as 0 and 1, respectively, the above equation can be further simplified as:

$$D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) = \log \frac{1}{\sigma_{\hat{\mathcal{N}}}} + \frac{\sigma_{\hat{\mathcal{N}}}^2 + \mu_{\hat{\mathcal{N}}}^2}{2} - \frac{1}{2}. \tag{7.11}$$

The KL divergence measures how one probability distribution differs from another. A larger value of $D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N})$ indicates that the output distribution of the memristive NN is different from the expected distribution due to non-idealities in the parameters. Specifically, a threshold th can be defined, where $D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) \geq th$ indicates non-ideal parameters in the memristive NN. The specific choice of th depends on several factors, which will be discussed later in Section 7.2.3.

Please note that other distance functions, such as the Jensen-Shannon divergence (which is a symmetrized version of the KL divergence), can also be used. Alternatively, for simplicity, evaluating only the $\mu_{\hat{\mathcal{N}}}$ and $\sigma_{\hat{\mathcal{N}}}$ values is also sufficient for detecting faults and variations.

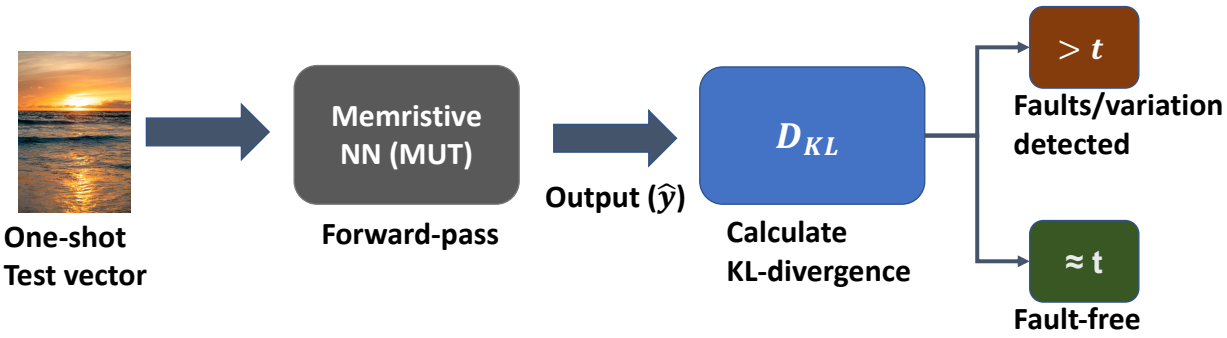


Figure 7.11.: Flow diagram of our proposed one-shot testing approach. A KL-divergence value greater than a predefined threshold indicates faults or variation in the memristive NN.

The test vector (stored in the hardware) can be applied periodically during the online operation, and the deviation from the expected distribution can be used as an indicator for faults and variation in the memristive NN. The general flow diagram of our one-shot testing approach is depicted in Fig. 7.11.

7.2.1.3. Hardware Needed to Compute The KL Divergence

The computation of the KL divergence involves a series of arithmetic operations, including logarithm, mean and standard deviation calculations, square root, addition, and subtraction (see Equation 7.11).

It is important to note that hardware acceleration for the KL divergence computation is not essential. This calculation can be effectively performed using software, as it is applied to the output of the neural network (following a black-box approach).

Alternatively, if the primary objective is to minimize the computational overhead associated with monitoring the status of the neural network, a simpler approach could be adopted. In this case, the mean and standard deviation (or variance) of the logit distribution can be observed. Therefore, the computational requirements can be significantly reduced. This approach would primarily involve summation, subtraction, multiplication, and division operations, all of which are typically supported by most AI accelerators.

7.2.1.4. Proposed test vector generation method

In order to make the proposed one-shot testing method possible, the distribution of logits $\mathbf{z}^L$ of a model should not only be standardized, but also be done with a single test vector and one forward-pass, i.e., *one-shot*. We generate a special test vector for this purpose with a specific learning objective that is unrelated to the original learning objective of the model. The original learning objective of the model remains unchanged, and no knowledge about the original learning process is required for the test vector generation process. In our approach, gradient descent is applied to the input (test vector) of a model rather than to the model parameters and variables. Therefore, the model parameters and variables remain unchanged. Consequently, the baseline accuracy of the model in the original task remained unchanged.

Learning objective Since our learning objective is to produce a standard Gaussian distribution for $\mathbf{z}^L$ by optimizing only the input to the model, several loss functions can be designed to encourage the $\mathbf{z}^L$ distribution $\hat{\mathcal{N}}$ to have a mean of 0 and a standard deviation of 1. For example:

$$\underset{\mu_{\mathcal{N}}\to 0,\sigma_{\mathcal{N}}\to 1}{\arg\min}\ \frac{1}{N}\sum_{i=1}^{N}\mathbf{z}_i^L \log\frac{\mathbf{z}_i^L}{\hat{\mathbf{z}}_i^L}, \tag{7.12}$$

minimizes pointwise KL-divergence loss between the logits $\mathbf{z}^L$ model and ground truth value $\hat{\mathbf{z}}^L$. Alternatively,

$$\underset{\mu_{\hat{\mathcal{N}}}\to 0,\sigma_{\hat{\mathcal{N}}}\to 1}{\arg\min}\ (\mu_{\hat{\mathcal{N}}})^2 + (1-\sigma_{\hat{\mathcal{N}}})^2, \tag{7.13}$$

encourages $\mu_{\hat{\mathcal{N}}}$ and $\sigma_{\hat{\mathcal{N}}}$ to be close to 0, and 1, respectively. Regression loss, such as

$$\underset{\mu_{\hat{\mathcal{N}}}\to 0,\sigma_{\hat{\mathcal{N}}}\to 1}{\arg\min}\ \frac{1}{C}\sum_{i=1}^{C}(\mathbf{z}_i^L - \hat{\mathbf{z}}_i^L)^2, \tag{7.14}$$

can also be used. Here, C denotes the number of output classes in the NN.

The ground truth $\hat{\mathbf{z}}^L$ for the training can be defined as

$$\hat{\mathbf{z}}^L = \frac{\mathbf{z}^L - \mu_{\hat{\mathcal{N}}}}{\sigma_{\hat{\mathcal{N}}}}, \tag{7.15}$$

or can be sampled from a unit Gaussian distribution. The number of samples should be the same as the number of output classes of an NN model. Our learning objective can be considered supervised learning, as we have target value $\hat{\mathbf{z}}^L$ for the objective.

The proposed one-shot test vector produces a standardized output distribution across different models and generated test vectors, as shown in the bottom half of Fig. 7.10(a) and (b). Furthermore, the relative deviation of the output distribution for the one-shot test vector is significantly higher, as demonstrated in Fig. 7.10(c). As a result, our one-shot test vector is considerably more sensitive to non-ideal parameters.

Initialization Let $\bar{\mathbf{x}}$ be the learnable one-shot testing vector with shape $[H, W, C_{in}]$ (assuming a colored image for image related tasks) that is optimized based on the loss function (7.11). Here, H, W, and C_{in} denote height, width, and number of channels, respectively. The correct initialization of $\bar{\mathbf{x}}$ is crucial to proper learning, where the initial values are assigned to each pixel of $\bar{\mathbf{x}}$ before training. The convergence speed and the final loss value depend greatly on proper initialization. Additionally, appropriate initialization is essential for deeper networks, as the gradient is propagated all the way back to the input.

We initialize $\bar{\mathbf{x}}$ element-wise with random values drawn from a unit Gaussian distribution as follows:

$$\bar{\mathbf{x}}_{H,W,C} \sim \mathcal{N}(0,1) \tag{7.16}$$

Element-wise initialization enables fine-grained control over the initialization process and is commonly used in deep learning.

Alternatively, initialization from out-of-distribution data, i.e., data that does not belong to the training set, also works well. This means that stock images from the Internet can also be used for initialization. Therefore, access to training data is still not necessary. Fig. 7.12 shows some examples of test vectors generated with their initial images. Our optimization procedure makes minute adjustments to the stock photos. Therefore, they are visually indistinguishable.

The overall algorithm for the proposed one-shot test vector generation is summarized in Algorithm 4. To accelerate the learning process, we propose optimizing the test vector with an exponential decay learning rate for every $\mathcal{K}$ iteration.

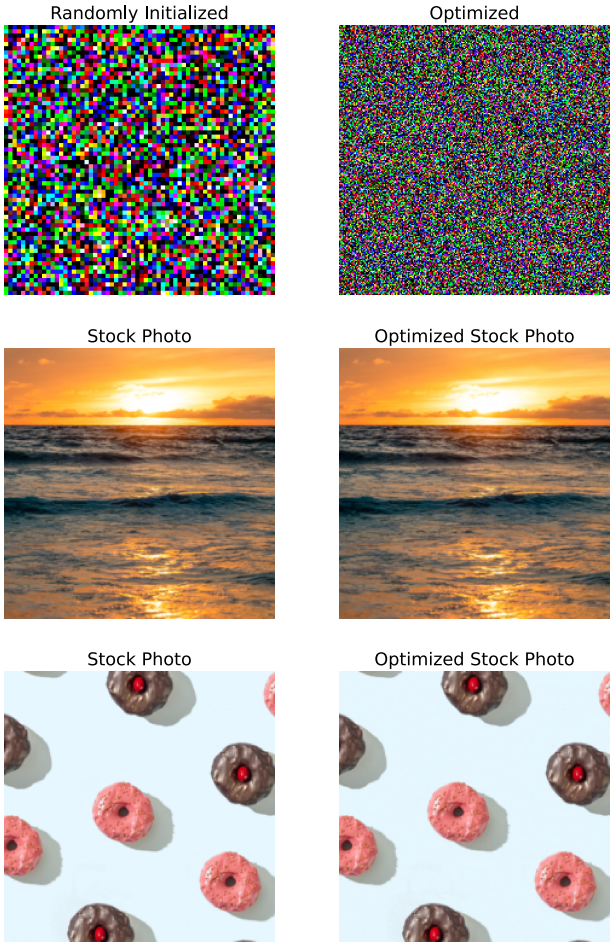


Figure 7.12.: Some examples of the proposed one-shot test vector for the DenseNet-121 topology. To the naked eye, optimized stock images appear identical to their original images. Nevertheless, they differ marginally.

Algorithm 4 One-shot test vector generation using gradient descent with an exponential decaying learning rate

Require: Pre-trained network $\mathcal{F}$, loss function $\mathcal{L}(.)$, initial learning rate α_0, number of iterations $\mathcal{E}$, Decay rate $\mathcal{K}$, and shape of the test vector $\bar{\mathbf{x}}$ $[H, W, C_{in}]$.
Ensure: One-shot test vector $\bar{\mathbf{x}}$
Initialize $\bar{\mathbf{x}}$ element-wise with random values from a
unit Gaussian distribution
for $e = 1 \ldots \mathcal{E}$ **do**
    Perform forward pass through $\mathcal{F}$ with input $\bar{\mathbf{x}}$ to obtain logits $\mathbf{z}^L$
    Compute loss $\mathcal{L}(\mathbf{z}^L, \hat{\mathbf{z}}^L)$
    Calculate gradient $\nabla\mathcal{L}$ with respect to $\bar{\mathbf{x}}$
    Compute the current learning rate α_t:

$$\alpha_t = \begin{cases} \alpha_0 & \text{if } t \bmod \mathcal{K} \neq 0 \\ \alpha_{t-1}/10 & \text{if } t \bmod \mathcal{K} = 0 \end{cases}$$

    Update $\bar{\mathbf{x}}$ using gradient descent with the current learning rate: $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} - \alpha_t \nabla\mathcal{L}$
end for

7.2.2. Relevance of Normalization Methods for Standardizing the Output Distribution

To reiterate, normalization methods such as batch normalization standardize neuron activations before applying affine transformations. However, they are not suitable for our proposed one-shot testing method due to the following reasons:

a) Conventional normalization methods are typically applied to intermediate activations of a NN and are not designed to directly standardize the output distribution, which is the primary goal of our one-shot testing method. **b)** Batch normalization, as an example, requires multiple test vectors (batch of inputs) to estimate the mean and variance of the distribution, conflicting with the one-shot nature of our method, which relies on a single test vector for output distribution standardization. Although other normalization techniques, such as group normalization [53], have been proposed for small batch sizes, they are designed for specific tasks such as sequence-to-sequence learning, recurrent neural networks (RNNs), or style transfer, and may not be directly applicable or easily adaptable to all deep learning tasks. **c)** Finally, normalizing the model output may necessitate retraining the model using the entire training dataset, which could be computationally expensive, require access to the training data, and potentially negatively impact the model's performance, as it may not generalize well to unseen data.

In general, the normalization method, in this case, would process the output of the MUT to produce the standardized output. Due to this processing, there is a risk that some fault-masking may occur. Thus, our unique approach to standardizing the output distribution aligns well with the requirements and objectives of our one-shot testing method.

7.2.3. Simulation Results

7.2.3.1. Fault Modelling and Injection Framework

Modelling Conductance Variations Conductance variations are only subject to memristive devices. Memristive technology and external environmental conditions influence conductance variation during online operations and in the manufacturing process. In this section, we employ the conductance variation model proposed in [41] that considers both the device-to-device manufacturing process variations (spatial fluctuation) and the thermal variations (temporal fluctuation during online operation). Since the distribution of memristive variations changes from one memristor technology to another, the variation model injects two types of variations into NN weights: multiplicative and additive Gaussian noise. Both types of variation are injected into the weight matrix of all layers as random noise, with a noise scale of η_0 used to control the severity of the variations. For each fault run, a different random sample is taken from the variation model. Furthermore, since various noise scales of η_0 are used, our evaluation considers the various degrees of process and thermal variations of the memristive cells.

For variations in the MAC results, we have modeled process and run-time variations of memristors affecting the MAC results as Gaussian noise. Thus, we inject Gaussian noise (with varying σ) into the MAC results of hidden layers.

Modelling Online and Manufacturing Faults In terms of online and manufacturing faults, we consider the fault model $f(\cdot)$, which takes into account all the common faults discussed in Section 2.10. Since the ultimate effect of various types of fault is to flip the affected memory cell from its desired level to another, the fault model $f(\cdot)$ considers the flipping fault model. Note that although in stuck-at faults, a memory cell is stuck at either a low or high state, its effect is seen only if the faulty cell is stuck at a level opposite to the desired level. Therefore, flipping fault models are relevant for stuck-at-faults as well.

Also, we consider two different kinds of flipping fault models depending on the mapping employed or hardware accelerator architecture: bit-wise and level-wise. NN parameters can be encoded bit-wise, with

eight memory cells representing a single parameter. Our bit-wise fault model targets this kind of parameter encoding and can be expressed as:

$$W_{flip} = f(\mathcal{P}_{flip}, W_{orig}) \tag{7.17}$$

Here, $\mathcal{P}_{flip}$ and $f(\cdot)$ represent the percentage of injected bit-flip faults and the fault model function, respectively. Specifically, the fault model $f(\cdot)$ randomly samples $\mathcal{P}_{flip}\%$ of the bits of weights in each layer and flips their bits from 1 to 0 and vice versa. On the other hand, for parameter mapping with multi-level memristive cells in a memristive NN accelerator, the (level-wise) fault model $f(\cdot)$ randomly sets the weights to a value between -127 and 127.

Furthermore, as previously stated, the activations of NNs are also susceptible to permanent and soft faults. We model them as stuck-at low/high faults and random flips. For stuck-at-low/high, we define defective ReLU (dReLU), where the ReLU activation output can be stuck at a zero value simulating stuck-at-low faults and a high value such as 10^2 for stuck-at-high faults. On the other hand, we define random-flip ReLU (rfReLU) to simulate soft faults modeled as random-flips. In rfReLU, the ReLU activation output is randomly set to a value between 0 and 10^2 in each forward pass.

Note that multiple faults are considered in a single memory array. However, single and multiple faults are considered in the intermediate activations of the NN. Additionally, the distribution of faults in the memory array is randomly chosen for each Monte Carlo fault simulation run. Therefore, an evaluation of various possible faults that occur during both the online and manufacturing processes is performed.

7.2.3.2. Simulation Setup

In this section, we have abstracted circuit-level details and evaluated our proposed one-shot approach using PyTorch-based simulation. We target *hard-to-detect* deviations in NN accelerators. As the name suggests, detecting these kinds of deviations is difficult, as they cause subtle changes in the output distribution and, in turn, inference accuracy. In contrast, we have found that a large change in accuracy correlates with a large relative shift in the output distribution and is easier to detect with our approach compared to others.

Furthermore, instead of a simpler dataset like MNIST, **we evaluated our method on larger scale pre-trained topologies, with up to 201 layers, trained on the more challenging ImageNet** dataset [8], which is a large-scale image recognition dataset with 1000 classes, approximately 13 million training data points, and 50,000 validation data points. Furthermore, we tested our approach on popular semantic segmentation topologies trained on real-world brain MRI datasets and Microsoft's COCO benchmark dataset [216, 217]. Semantic segmentation is considerably more challenging than image classification since it involves assigning labels to individual pixels in an image. Table 7.3 summarizes all pre-trained models evaluated, their accuracy, and the number of parameters. All pre-trained models are accessible through the PyTorch Hub. Also, *parameters and variables of the models were not modified* for the proposed test vector generation process or during online operation, as our test vector generation process is a black-box approach. Furthermore, no knowledge or modification of the training process for the models is required. Therefore, the original task and the accuracy of each model under test remain unchanged, and the accuracy of each mode is shown in Table 7.3.

For the fault coverage analysis, we have done the Monte Carlo simulation to simulate the effect of per-chip and online variations, as well as various faults modeled as bit-flip. Specifically, 1000 chip instances are evaluated for each noise level for variations and fault percentages. For the evaluation of each chip instance, the single-shot test vector is forward-passed through the model once. One and only one test vector is generated using the loss function equation 7.9 for each of the models shown in Table 7.3. The choice of this loss function was driven by its ability to provide a clear indication of the KL divergence value of the fault-free NN. Monitoring the loss value during training offers a transparent view of the trend in the KL

Table 7.3.: Showing the evaluated (pre-trained) models for classification and semantic segmentation tasks, along with their respective information such as inference accuracy, number of parameters, layers, and dataset used for training.

Model	Classification			
	Inference Acc.	Parameters	Layers	Dataset
ResNet-18 [9]	69.76%	11.7×10^6	18	Imagenet [8]
ResNet-50 [9]	76.13%	25.6×10^6	50	
ResNet-101 [9]	81.89%	44.5×10^6	101	
DenseNet-121 [218]	74.43%	8×10^6	121	
DenseNet-201 [218]	76.89%	20×10^6	201	
MobileNet-V2 [219]	71.87%	3.5×10^6	52	
	Semantic Segmentation			
	Pixelwise Acc.	Parameters	Layers	Dataset
U-Net [164]	98.75%	7.76×10^6	23	Brain MRI [216]
DeepLab-V3 [220]	91.2%	11.03×10^6	72	COCO [217]

divergence, facilitating an efficient training process. One of the key advantages of using the Gaussian KL Loss is its utility in early stopping the training process. That is, once we have obtained a KL-divergence value that is sufficiently close to zero, the training can be stopped.

Note that each NN accelerator is evaluated using the proposed single-shot test, and the respective output distribution for evaluation is generated using a single forward pass. Monte Carlo simulation is performed only for the evaluation of different crossbars, faults, and variation distributions. In a real-world scenario, *only a single forward-pass on the hardware is required.*

We report fault coverage as the ratio between detected faults ($D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) \geq \mathrm{th}$) and overall fault runs ($\mathcal{R}$) and can be described as

$$\text{fault coverage} = \frac{\#\text{ of } D_{\mathrm{KL}}(\hat{\mathcal{N}} \parallel \mathcal{N}) \geq \mathrm{th}}{\mathcal{R}} \times 100. \tag{7.18}$$

Here, $\mathcal{R}$ refers to the number of fault runs for our Monte Carlo fault simulation. We specifically target non-benign faults in neural networks, which noticeably degrade accuracy. However, the fault and variation rates are chosen in such a way that the accuracy degradation is marginal for the lowest fault rate, and with increasing fault rate, the accuracy deteriorates gradually.

As a result, all the reported fault coverages are true positive rates (TPR) which states the number of non-benign faults impacting the accuracy that are correctly detected. On the other hand, the false negative rate (FNR) can be obtained from equation 100 − fault coverage. FNR tells the number of instances that NN is classified as not faulty (having performance degradation), but in reality, they are faulty. Ideally, FNR should be 0% and TPR or fault coverage should be 100%.

Single or multiple faults that do not degrade accuracy are defined as benign faults. This is because NN is inherently fault-tolerant up to a certain degree and does not significantly influence the logit distribution or inference accuracy of the model. Benign faults are not the target of this section.

Note that although we inject variation and faults into all parameters, ReLU activations, and MAC results, our fault coverage does not necessarily imply the detection of all possible faults that may occur.

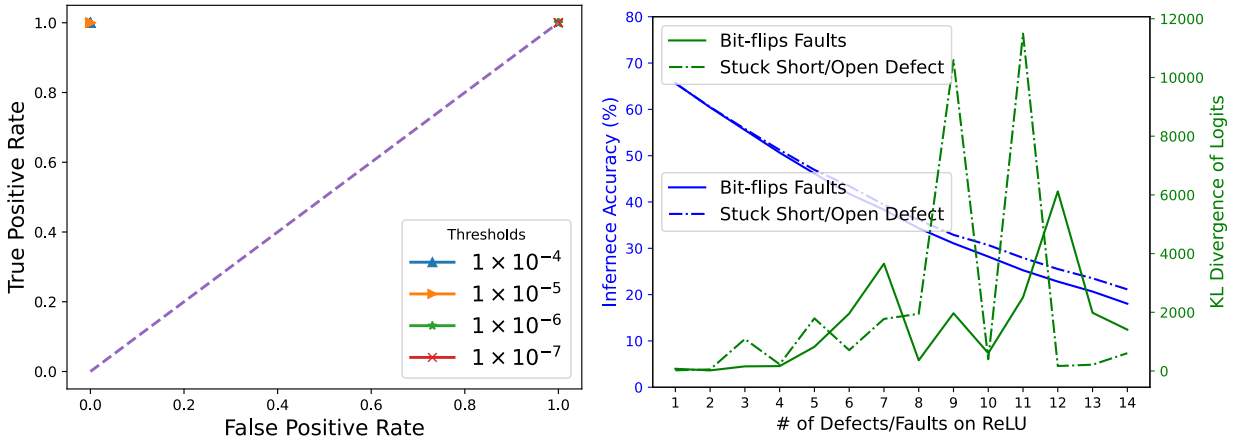


Figure 7.13.: (Left) Receiver operating curve (ROC) of proposed fault detection method and (Right) change in accuracy and KL-Divergence value with fault rates for the Movilenet-V2 model and Imagenet dataset. **It is recommended to view this figure in color.**

7.2.3.3. Detecting Variations in a One-Shot

For classification tasks using the ImageNet dataset, Table 7.4 evaluates the fault coverage achieved by a proposed one-shot method on multiplicative and additive variations. The six SOTA models consistently achieve 100% fault coverage across various noise scales (η_0). Our results indicate the robustness of the one-shot method in adapting to diverse levels of noise.

Similarly, for semantic segmentation tasks, as shown in Table 7.5, the proposed one-shot method on multiplicative and additive variations with a range of noise scales (η_0) consistently achieves 100% fault coverage. This further underscores the robustness of our one-shot method across different tasks.

Table 7.4.: The fault coverage (%) achieved by the proposed one-shot method on multiplicative and additive variations with different noise scales η_0. All the models are trained on the ImageNet dataset.

Model	Multiplicative Variations						Additive Variations				
	$\eta_0 = 0.01$	$\eta_0 = 0.02$	$\eta_0 = 0.04$	$\eta_0 = 0.06$	$\eta_0 = 0.08$	$\eta_0 = 0.10$	$\eta_0 = 0.0001$	$\eta_0 = 0.0002$	$\eta_0 = 0.00025$	$\eta_0 = 0.0003$	$\eta_0 = 0.00035$
ResNet-18	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-50	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-101	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DenseNet-121	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DenseNet-201	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
MobileNet-V2	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

7.2.3.4. Detecting Faults in a One-Shot

For ImageNet classification in various SOTA topologies, Table 7.8 demonstrates a similarly high level of fault coverage under both bit-flip and level-flip fault conditions. For each model, as the fault rate increases, the percentage of fault coverage generally improves. At higher fault rates, such as 0.05% and 0.1%, most models achieved 100% fault coverage. At lower fault rates, the shift in the output distribution is very low, resulting in a few false-negative cases. However, by reducing the threshold to a value closer to the KL divergence value in a fault-free model, the number of false negative cases can be reduced (see Table 7.6). Nevertheless, our results indicate the resilience of our one-shot approach to various types of faults at different rates.

Table 7.5.: Fault coverage of semantic segmentation models utilizing the proposed one-shot testing method. The models are evaluated under multiplicative and additive variations with different noise scales η_0 to underscore the robustness of the models across different scenarios.

Model	Additive Variations					
	$\eta_0 = 0.00002$	$\eta_0 = 0.00004$	$\eta_0 = 0.00006$	$\eta_0 = 0.00008$	$\eta_0 = 0.00010$	$\eta_0 = 0.00012$
U-Net	100%	100%	100%	100%	100%	100%
	$\eta_0 = 0.02$	$\eta_0 = 0.04$	$\eta_0 = 0.06$	$\eta_0 = 0.08$	$\eta_0 = 0.1$	$\eta_0 = 0.12$
DeepLab-V3	100%	100%	100%	100%	100%	100%
	Multiplicative Variations					
	$\eta_0 = 0.01$	$\eta_0 = 0.02$	$\eta_0 = 0.04$	$\eta_0 = 0.06$	$\eta_0 = 0.08$	$\eta_0 = 0.08$
U-Net	100%	99.9%	100%	100%	100%	100%
	$\eta_0 = 0.01$	$\eta_0 = 0.02$	$\eta_0 = 0.04$	$\eta_0 = 0.06$	$\eta_0 = 0.08$	$\eta_0 = 0.08$
DeepLab-V3	100%	100%	100%	100%	100%	100%

7.2.3.5. Detecting Faults and Variations at Activation in a One-Shot

Similarly, our method demonstrates remarkable efficacy in identifying both single and multiple faults, including stuck-at-low, stuck-at-high, and bit-flip faults, in NN activations, as detailed in Table 7.10. As can be seen, on a range of SOTA ImageNet models, it consistently achieves a fault coverage of 100%. Furthermore, our method is equally effective in detecting manufacturing variations affecting MAC values in hidden layers, maintaining a 100% fault coverage. This uniform success across various models and fault types underscores the robustness and reliability of our proposed method for detecting faults and variations in NNs.

7.2.3.6. Analysis of False Positives

While the true positive rate, or fault coverage, is important, the false positive rate (FPR) is also important to assess the effectiveness of a testing approach. FPR reports instances where the NN is incorrectly identified as faulty, when, in reality, there is no fault in the NN, that is, a false alarm. Since our approach is an explicit or pause-and-test method of testing with a specifically generated single test vector, classification is always binary (0, 1). Consequently, the FPR for all models is summarized in Table 7.7. As can be seen, our approach has an FPR of zero, which means that false alarms are never raised. However, there is a relationship between FPR, TPR, and the chosen threshold for fault detection. The receiver characteristic curve (ROC) for the MobileNet-V2 topology is shown in Figure 7.13. It can be depicted that our approach is exceptionally effective at detecting faults/variations with increasing fault rates when the threshold is set correctly. Since the baseline KL divergence value is $1.56182795763015753 \times 10^{-6}$ (see Table 7.12), if the threshold is below the baseline KL divergence value, then both the TPR and the FPR become one. This suggests that the proposed method can identify all fault instances correctly and raise false alarms at these thresholds. On the other hand, the proposed method achieves a perfect fault classification with a TPR of 1 and an FPR of 0 when the chosen threshold is above the baseline. This point is considered the optimal point on an ROC curve, representing 100% sensibility (no false negatives) and 100% specificity (no false positives). We observed the same for other models.

Thus, the general relationship between TPR and FPR is given by the threshold value chosen for faults/variations detection. As shown in Table 7.6 as the threshold is reduced coverage or the TPR increases, but the ROC curve suggests that it also increases the FPR. **As a result, a threshold should be chosen that is greater than or equal to the baseline KL divergence value. It should never be below the baseline, as it will raise false-positive alarms.**

Similarly, our proposed method can achieve high fault coverage in both bit-flip and level-flip fault conditions for semantic segmentation tasks on two SOTA topologies, as demonstrated in Table 7.9. The trend in

Table 7.6.: The effect of threshold t on false-negative test cases. As the threshold is reduced, the fault coverage increases.

Threshold	% of faults		
	$\eta_0 = 0.02$	$\eta_0 = 0.025$	$\eta_0 = 0.33$
1×10^{-4}	98.7%	99.2%	99.1%
1×10^{-5}	99.4%	99.8%	99.8%
1×10^{-6}	99.9%	99.8%	100%
1×10^{-7}	100%	100%	100%

Table 7.7.: Evaluation of false positive rate (FPR) of the proposed approach on different topologies when NN is fault or variation-free. Since FPR is evaluated on a single test vector, it is represented as a binary (0, 1) value with 0 representing no false positive (ideal scenario) and 1 representing a false positive classification.

	Topologies							
	ResNet-18	ResNet-50	ResNet-101	DenseNet-121	DenseNet-201	MobileNet-V2	U-Net	DeepLab-V3
FPR	0	0	0	0	0	0	0	0

fault coverage percentage for each model is similar to that of the models used for ImageNet classification. We also found that lowering the threshold can have a similar effect on fault coverage, as observed in the ImageNet classification models.

Table 7.8.: Evaluation of the fault coverage of SOTA ImageNet classification models under different fault scenarios, including Bit-flip and Level-flip faults, and varying fault rates.

Model	Bit-flip					Level-flip				
	% of faults					% of faults				
	0.02%	0.025%	0.033%	0.05%	0.1%	0.02%	0.025%	0.033%	0.05%	0.1%
ResNet-18	98.6%	99.3%	99.6%	100%	100%	99.9%	99.9%	100%	100%	100%
ResNet-50	99.6%	99.9%	100%	99.9%	100%	100%	100%	100%	100%	100%
ResNet-101	98.9%	99.6%	99.5%	99.9%	100%	99.7%	100%	100%	100%	100%
DenseNet-121	99.9%	99.6%	100%	100%	100%	99.9%	100%	99.9%	100%	100%
DenseNet-201	99.1%	99.7%	99.8%	100%	100%	99.1%	100%	100%	100%	100%
MobileNet-V2	99.8%	99.8%	100%	100%	100%	99.4%	99.8%	99.8%	100%	100%

Table 7.9.: Fault coverage performance of semantic segmentation models U-Net and DeepLab-V3 under Bit-flip and Level-flip fault scenarios with varying fault rates.

Model	Bit-flip					Level-flip				
	% of faults					% of faults				
	0.02%	0.025%	0.033%	0.05%	0.1%	0.02%	0.025%	0.033%	0.05%	0.1%
U-Net	100%	100%	100%	100%	100%	99.9%	99.9%	100%	100%	100%
DeepLab-V3	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

7.2.3.7. Comparison with State of the Art and Overhead Analysis

Our proposed method is compared against the related works that uses the functional test generation method and focuses on the compaction of the test pattern. With only one test vector and test query, the proposed one-shot testing method outperforms existing methods [37], [117], [118], and [41] in all metrics listed in Table 7.11. Therefore, the proposed method requires significantly fewer test vectors and queries compared to other methods. Furthermore, the proposed approach consistently achieves 100% fault coverage, outperforming methods [37, 117, 118] that range from 76% to 99.27% coverage. Additionally, the proposed method is the most memory efficient, requiring only 0.012288 MB, which is much lower than the

Table 7.10.: The analysis of the effectiveness of our method in detecting single and multiple faults (stuck-at low/high) and bit-flip faults in ReLU activations, and manufacturing variations affecting the MAC values of the hidden layer, for state-of-the-art ImageNet classification models.

Topology	# of Stuck-At Low /High Faults on ReLU					# of Bit-flip Faults on ReLU					Manufacturing Variation (σ_{var})				
	1	2	3	4	5	1	2	3	4	5	0.001	0.002	0.002	0.004	0.005
ResNet-18	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-50	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-101	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DenseNet-121	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DenseNet-201	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
MobileNet-V2	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

other methods, regardless of whether re-training data is stored in hardware or not. Moreover, our method does not rely on storing re-training data in hardware to reduce memory consumption, unlike the methods proposed by [37], and [41].

The test application time (latency) and test energy are directly proportional to the number of test vectors used for testing. For example, the testing method [117] requires 1024 test vectors. Therefore, their method requires 1024× more MAC operations and power consumption. In our comparisons, we assume that the hardware implementation, NN topology, and NVM technology are the same.

The analysis presented in Table 7.11 is based on the numbers reported in related works. To calculate memory consumption, we utilized the bit-width reported in [37] for images and test labels. Note that our approach does not require the storage of any labels.

Table 7.11.: Compares the proposed approach with the existing methods using four performance metrics: fault coverage, memory storage overhead (with and without re-training data), number of test queries required, and fault-detection resolution. To ensure a fair comparison, the analysis of our approach is conducted on the CIFAR-10 dataset.

	[37]	[117]	[118]	[41]	Proposed
Size of test vectors	10000	1024	10-50	16-64	**1**
# of test queries (normalized)	10000	1024	10-50	17	**1**
Memory overhead (MB)	0.015[1]	234.42[1]	0.154[1]	0.000256[1]	**0.012288**[1]
	245.775[2]	234.42[2]	0.154[2]	0.1966[2]	**0.012288**[2]
Coverage (%)	99.27[3]	98[3]	76[3]/ 84[4]	100[4]	**100**[3]

[1] Re-training data is stored in hardware.
[2] Re-training data is not stored in hardware.
[3] Synthetic testing data
[4] Original training data used as testing data

7.2.4. Discussion and Future Works

Guaranteed Single Shot Test Vector The proposed one-shot test method determines the logit distribution of a model by fitting it to a Gaussian distribution. The population size of the logit distribution is determined by the number of neurons in the last layer (output classes) of the NN. In cases where the number of neurons in the last layer of the NN is greater than 20, we can consistently find many test vectors for each model that produce a logit distribution closely resembling a unit Gaussian. Thus, for an NN with a large enough number of classes, our method guarantees that there exists a test vector that produces a standardized normal distribution for the NN logits. Table 7.12 summarizes the mean, standard deviation, and KL divergence value of the logit distribution of the generated test vectors for each model that was evaluated. We consider the mean and standard deviation obtained to be sufficiently close to zero and one, respectively. Also, the KL divergence value can be considered sufficiently close to zero, such that the logit distributions of the model are approximately unit Gaussian.

Approach for smaller number of classes in the last layer To avoid biased estimation of mean and standard deviation, it is important to have a sufficiently large number of output classes, such as 20 or more. For cases with fewer output classes, such as 2 or 5, our approach may not work well, as it becomes harder to obtain a unit Gaussian logit distribution. For these models, alternative statistical methods, such as Bayesian approaches, could be considered, but they are beyond the scope of this section. Additionally, the number of classes in the last layer can be increased by adding proxy neurons at the expense of a higher memory overhead. However, in this case, the method becomes an invasive method and may not be applicable to a pre-trained model. Alternatively, a best-case scenario is in which the test vector is optimized for a KL divergence value as close to zero as possible. Then, by storing this KL divergence value along with the test vector, even a model with a smaller number of classes can be tested.

Even though we have standardized the logit distribution of each model to a unit Gaussian, it is not necessary to do so. In future work, we would like to explore other Gaussian or non-Gaussian distributions that may be more generalizable across a range of output classes of a model.

Precision of The Test Vector Input of the NN, such as an image or video, is typically in the 32-bit floating-point format during inference. Although inference data does not require storage, test vectors, which are crucial in our methodology, do. Hence, their storage overhead, although negligible, can be further reduced by reducing their bit precision.

The proposed one-shot testing vectors consider full precision as bit precision (32-bit floating point) for two reasons: optimization accuracy and quantization impact. Since we perform gradient descent on a single input image, keeping it at 32-bit full precision helps in fine-grained optimization during the gradient descent process of the test vector. On the other hand, reducing the bit-precision of the test vector can lead to an increased KL divergence value from the baseline due to quantization errors. Such errors can adversely affect our fault coverage, as they can mask or mimic the effects of actual faults.

Fault Propagation and Their Impact In terms of memristive AI accelerators, the weight matrices of each layer are mapped to different crossbar arrays with memristive devices at each crosspoint. In the event of faults or variations in the crossbar of a layer, these faults would indeed affect subsequent layers, as they receive faulty values as input. This is because the MAC result of the faulty layer will be incorrect, resulting in the following layers receiving incorrect input. However, the location of these faults would vary between crossbars and layers, which can be referred to as device-to-device variation. Additionally, for different chips that map the same neural network topology, the fault locations on the crossbars are expected to vary, reflecting chip-to-chip variations. Note that in terms of conductance variation, this means that the conductance value of each memristive device varies from device to device and chip to chip.

We have modeled both of these scenarios in our fault model and fault injection runs. We randomly assigned fault locations for each layer and for different runs to simulate these variations. That is, fault locations vary across different layers in a single fault run to simulate device-to-device variations. For each fault run, the fault locations are randomly chosen to simulate chip-to-chip variations.

To simulate the impact of different crossbars of different dimensions and manufacturing processes, we have considered different fault rates. Fault rates range from marginal to large, simulating precise and imprecise fabrication processes.

In systolic arrays and FPGA-based accelerators, permanent faults that affect the MAC unit likely affect all layers. However, this is not the case for memristive AI accelerators, as weights are mapped to different crossbars. In general, the issue of shared fault impacts across layers is not considered in our experiments. However, for these types of fault, our approach is equally efficient in detecting them. We have performed a small ablation study on the MobileNet-V2 topology, where the fault location for the level-flip of each layer remains the same to simulate the shared fault issue. We have found that in this case, our approach can indeed achieve 100% fault coverage.

Table 7.12.: Summary KL-Divergence, mean, and standard deviation of the logit distributions of the respective test vectors on evaluated topologies.

Topology	KL-Divergence	Mean	Standard Deviation
ResNet-18	$5.721813067793846 \times 10^{-7}$	0.0010	1.0001
ResNet-50	$1.0050367563962936 \times 10^{-5}$	0.0014	1.0030
ResNet-101	$1.466134563088417 \times 10^{-6}$	0.0009	1.0009
DenseNet-121	$3.338936949148774 \times 10^{-7}$	0.0008	1.0001
DenseNet-201	$1.5618279576301575 \times 10^{-6}$	0.0016	1.0003
MobileNet-V2	$1.5618279576301575 \times 10^{-6}$	0.0004	0.9997
DeepLab-V3	$1.0870280675590038 \times 10^{-8}$	0.0001	0.9999
UNet	3.8781×10^{-9}	0.00002	0.9999

Table 7.13.: Impact of faults and variation on the accuracy of the ImageNet models. Here, the precision shows the impact in relation to the smallest fault and variation rates.

Topology	Non-idealities Insertion Locations and Types						
	Weights				ReLU Activations		MACs
	Additive Variations	Multiplicative Variations	Bit-flips	Level-flips	# of Stuck-at Low/High Faults	# of Bit-flip Faults	Variation (σ_{var})
ResNet-18	69.50%	69.48%	68.88%	67.452%	67.80%	67.98%	68.74%
ResNet-50	75.83%	75.79%	72.36%	73.20%	72.33%	73.47%	75.03%
ResNet-101	77.21%	76.86%	76.22%	74.55%	67.70%	70.45%	71.00%
DenseNet-121	74.24%	73.93%	72.00%	71.41%	55.68%	55.05%	72.60%
DenseNet-201	76.59%	76.51%	59.924%	75.45%	46.38%	44.33%	71.35%
MobileNet-V2	71.14%	69.90%	70.378%	68.694%	65.65%	65.62%	70.14%

Fault Detection Granularity **Our method does not need a considerable number of faults or variations to be able to detect.** Thus, our approach can detect faults or variations with a graceful degradation in accuracy. As stated in Section 7.2.3.2, the fault rate is carefully chosen so that the accuracy degradation is marginal. For example, we have chosen bit flips and level flip rates of 0.02% to 0.1%. This means that 99.98% to 99.9% of the memories that store NN weights are fault-free.

We have performed our evaluation on large models with up to 201 layers and 44.5×10^6 parameters, and for each fault rate and model, we have performed 1000 Monte Carlo runs on Nvidia RTX 3080 GPUs. In total, we have conducted 260000 fault injection campaigns. Due to computational budget issues, we report an estimate of accuracy degradation on ImageNet models with the lowest fault/variation rates. Table 7.13 summarized the drop in accuracy due to various faults and variations that affect the weights and activation of ImageNet models. With an increasing fault rate, the accuracy gradually degrades from that point. As can be seen, in most cases, the drop in accuracy is marginal. For example, ResNet-18 has an accuracy drop of only 0.26%. In Figure 7.13 (right), we show a gradual change in accuracy and their respective KL-divergence values for the MobileNet-V2 model. Note that although the KL divergence value fluctuates due to the stochastic nature of the fault locations and magnitudes, the overall trend shows an increase in the KL divergence with increasing fault rate.

Testing Spinking Neural Netwroks We hypothesize that our approach can be extended to test spiking neural networks, as it is a black-box approach and a hardware-agnostic approach with a specifically designed test vector for each model. We consider that the fault models of SNNs are different compared to our fault model used in this section.

7.2.5. Scientific Impact of This Work

We outline the scientific impact of this work as follows:

1. **Single-Shot Testing:** Testing edge AI accelerator in single-shot allows for the efficient detection of faults and variations. Our approach significantly reduces the testing time and computational overhead compared to traditional methods, making it practical for real-time applications and continuous monitoring.
2. **Scalability:** Our approach can test even extreme-edge AI applications since the cost of testing is significantly low. Also, our approach is scalable to any edge AI accelerator architectures, NN topology, dataset, and tasks. Making it significantly attractive for diverse tasks.
3. **Distribution Monitoring:** Our work proposed to monitor the distribution of the output rather than model prediction, label, or accuracy. Consequently, our approach leads to a significant reduction in testing costs. Therefore, research in this direction is attractive for reducing testing costs.

7.2.6. Section Conclusion

In this section, we have introduced one-shot testing and a respective test generation method to test hardware accelerators for deep learning models based on memristor crossbars with a single test vector. Our method hypothesizes that memristive non-idealities correlate to changes in output distribution, and our testing method aims to detect this distribution shift with a single testing vector. The proposed approach demonstrates superior performance in fault coverage, memory storage overhead, and the number of test queries required, highlighting its effectiveness and efficiency compared to existing methods. Therefore, the method proposed in this section allows for significantly faster detection of faults and variations at a negligible overhead.

7.3. Few-Shot Testing Using Bayesian Test Vectors

Although the single-shot testing method proposed in the previous section is proven to be highly effective and low-cost, it has a drawback when it comes to testing NN with a low number of classes, e.g., a binary classifier. This is because they only produce a few elements in a single forward pass, which is insufficient to create a distribution.

This addresses this drawback with a Bayesian test vector approach and is based on our paper [221].

7.3.1. Problem Statement and Motivation

An NN with parameters θ gives the output y based on the input. Parameters θ are learned so that the predicted output y is close to ground truth $\hat{y}$. Therefore, even though one has access to in-distribution data, e.g., training or validation data, their respective output 1. would change from one input to the next, 2. one model to the next, and 3. uncertainty may not be minimum, see Fig. 7.14.

On the other hand, we have observed that the spread between logits of a NN (in general) increases as faults and variations in its parameter increase, as depicted in Fig. 7.15. Based on this observation, we propose to measure the standard deviation σ_y of the logits of a *model under test* (MUT) to estimate the *functional uncertainty of edge AI accelerator.*

To achieve the desired σ_y, we proposed a single Bayesian test vector generation framework. In a Bayesian test vector, each element of the input, e.g., pixels for classification tasks, is a distribution rather than a single-point estimate value, as shown in Fig. 7.17. As a result, one test vector can be learned, and $T = 1 \cdots T$ Monte Carlo samples (MC samples) can be taken from it. In each MC sampling step, element-wise sampling is performed, resulting in an input vector with a single value element. Each of the MC samples is forward passed T times through the MUT to create an output distribution for the calculation of uncertainty estimates σ_y (see Fig. 7.18). Here, T depends on the number of neurons C in the final layers. A MUT with a large

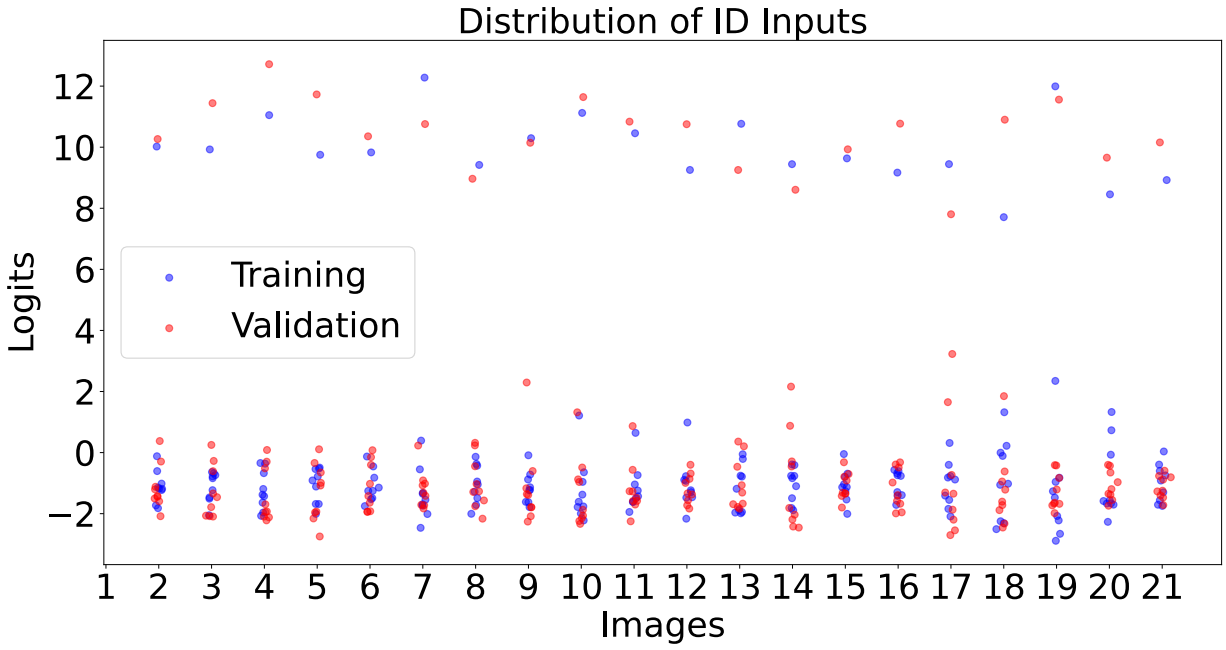


Figure 7.14.: Distribution of logits on the fault- and variation-free RepVGG [222] model trained on the CIFAR-10 dataset when several randomly sampled inputs from training and validation are applied. The distribution logits change from one input to another. **It is recommended to view this figure in color.**

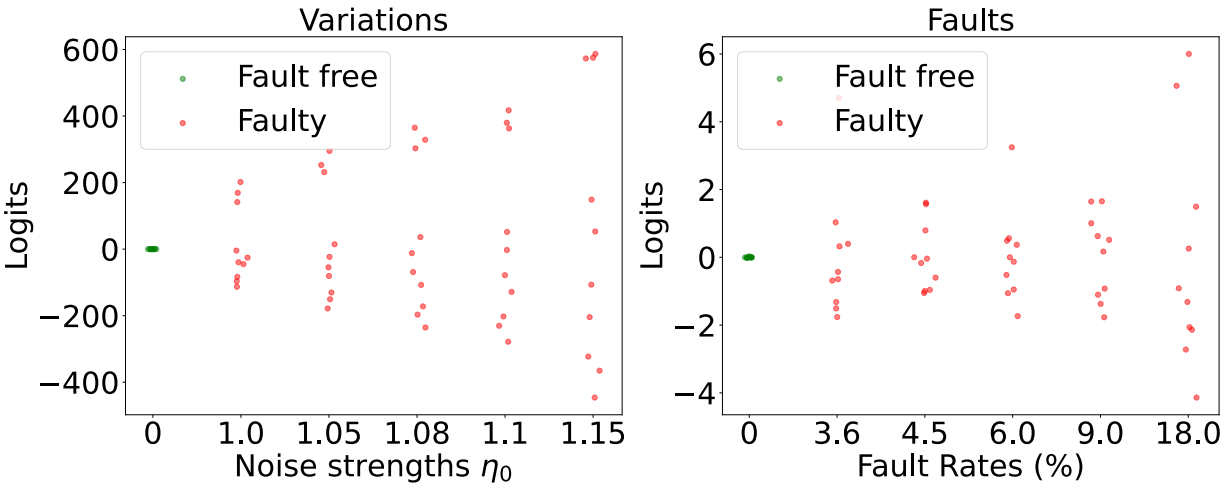


Figure 7.15.: Change in output distribution of logits of RepVGG [222] model on the CIFAR-10 dataset due to a) variations and b) faults. The spread among logits increases as the noise scale of variations and fault rate increases. **It is recommended to view this figure in color.**

C, for example, $C \geq 100$, has enough elements in the output vector to calculate the uncertainty with one sample and a forward pass. However, a model under test with a small C, for example, a regression or binary classification task with $C = 1$, will require multiple MC samples from the distribution of the Bayesian test vector and forward passes to calculate the functional uncertainty of the edge AI accelerator.

Our proposed Bayesian test vector can give a distinguished output in the case of functional uncertainty of the edge AI accelerator. In the case of a memristive NN that is fault-free and variation-free, in an ideal scenario, the uncertainty σ_y is minimum. Consequently, the uncertainty measures σ_y of a model given our proposed Bayesian test vector is much more sensitive compared to training and validation data, as demonstrated by Fig. 7.16. Therefore, our Bayesian test vector can potentially estimate uncertainty at low fault rates or variations.

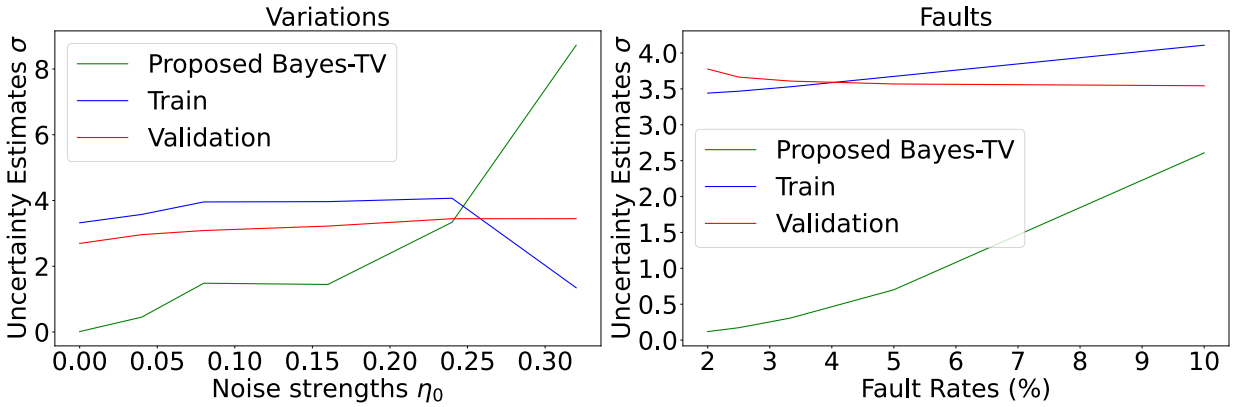


Figure 7.16.: Relative sensitivity of uncertainty estimates given proposed Bayesian test vector input as well as randomly sampled training and validation. The change in uncertainty estimates is much higher for our proposed Bayesian test vector. **It is recommended to view this figure in color.**

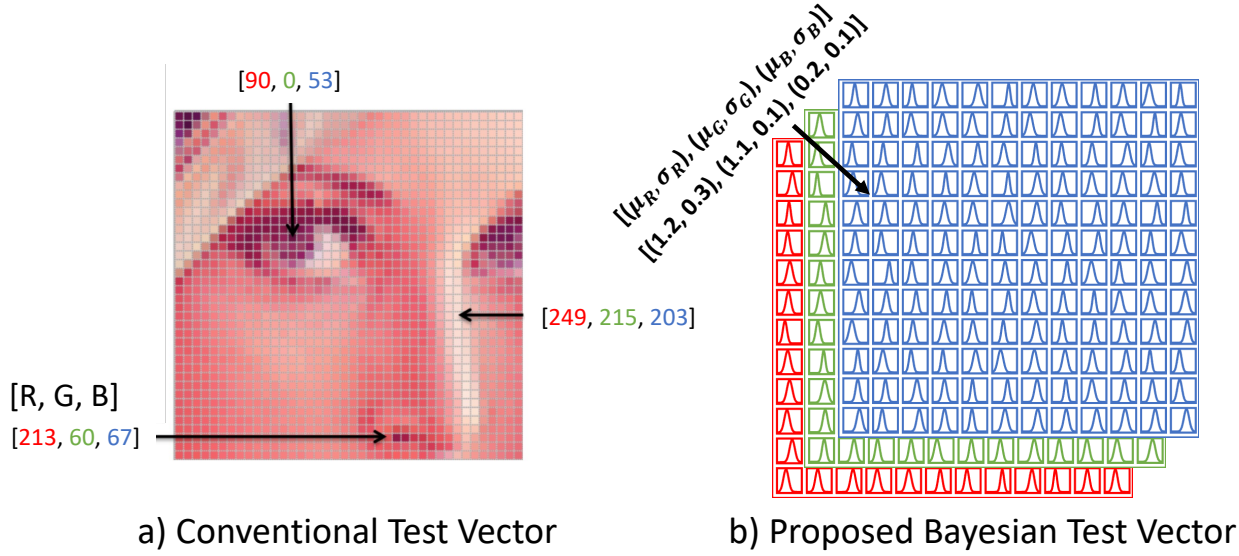


Figure 7.17.: An example of a) conventional test vector with each pixel representing *single point value* for the Red, Green, and Blue (RGB) channels, and b) proposed Bayesian test vector with each pixel representing *an independent distribution* for the RGB channels.

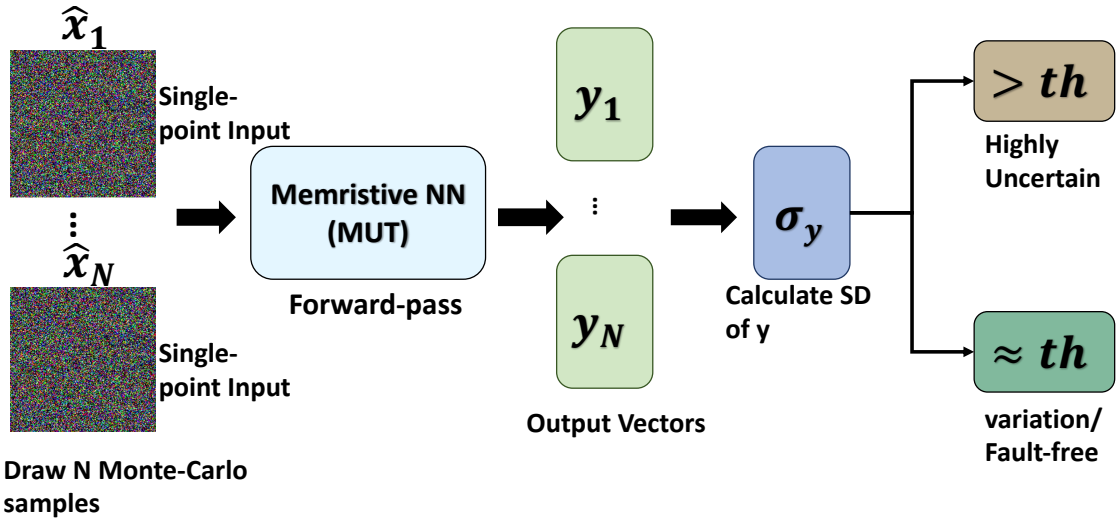


Figure 7.18.: Flowchart of the proposed uncertainty estimation method for a model under test (MUT). If the standard deviation (SD) of the output of MUT σ_y is higher than a pre-defined threshold th, then MUT is highly uncertain.

Although for a certain MUT, a few forward passes T are required to estimate the uncertainty, the overhead is still minimal. This is because, in deep learning, one of the factors that determines the difficulty of a task is the number of classes in it. A difficult task, for example, ImageNet-1k classification [8], usually requires a larger NN model, while an easier task requires a smaller NN model. In our approach, we take several samples T and forward-pass them through only on a small MUT. However, for a large MUT, we only take a single sample and forward-pass it through the MUT.

7.3.2. Methodology

7.3.2.1. Optimization Process of The Bayesian Test Vector

To obtain the test vector $\hat{x}$, we employ Bayesian inference. It uses Bayes' theorem to update our belief about the parameter of interest (in this case, the test vector $\hat{x}$) given some observed data (in this case, the output y). It can be written as:

$$P(\hat{x}|y) = \frac{P(y|\hat{x})P(\hat{x})}{P(y)}. \tag{7.19}$$

Where, $P(\hat{x}|y)$ is the posterior distribution of the test vector given the output, $P(y|\hat{x})$ is the likelihood of the output given the test vector, $P(\hat{x})$ is the prior distribution over the test vector, and $P(y)$ is the evidence or marginal likelihood of the output.

In our Bayesian optimization setting, the primary objective is to find the configuration of the test vector, $\hat{x}$, that maximizes the posterior distribution, $P(\hat{x}|y)$, given the output, y. However, since the direct computation of the posterior is often intractable, we resort to variational inference techniques to approximate our true posterior with a variational distribution, $q(\hat{x}|y)$, that comes from a simpler or more tractable family of distributions. Hence, we choose a Gaussian distribution as the variational distribution that is parameterized by mean $\mu_{\hat{x}}$ and standard deviation $\sigma_{\hat{x}}$.

Our objective can therefore be rephrased as finding the $\hat{x}$ that maximizes $q(\hat{x}|y)$, which can be considered as an approximation of the expected lower bound (ELBO). This is because, in the context of variational inference, the ELBO is the objective function that we aim to maximize. Hence, its negative can be seen as a loss function that needs to be minimized. The ELBO is defined as:

$$\text{ELBO} = \mathbb{E}_{q(\hat{x}|y)}[\log P(y|x)] - \text{KL}(q(\hat{x}|y)||P(\hat{x})) \tag{7.20}$$

Where the first term, $\mathbb{E}_{q(\hat{x}|y)}[\log P(y|x)]$, is the expected log-likelihood of the data to observe the desired output y, and the second term, $\text{KL}(q(\hat{x}|y)||P(\hat{x}))$, is the Kullback-Leibler (KL) divergence between the variational distribution and the prior, which can be thought of as a regularization term that makes sure that the variational distribution is close to the prior.

We have rephrased the objective function for our need as follows:

$$\text{Loss} = -\text{ELBO} = \sqrt{\frac{\sum (y - \mu_y)^2}{C}} + \alpha \times \text{KL}(q(\hat{x}|y)||P(\hat{x})) \tag{7.21}$$

Here, the term $\sqrt{\frac{\sum (y-\mu_y)^2}{C}}$ serves as a proxy for the likelihood term. It is designed to encourage the optimization process to find an $\hat{x}$ that results in y with as low a standard deviation as possible. This term essentially computes the root mean squared deviation of the outputs from their mean, reflecting our goal of minimizing output uncertainty. Furthermore, we introduce a hyperparameter α that can control the strength of the regularization term, $\text{KL}(q(\hat{x}|y)||P(\hat{x}))$.

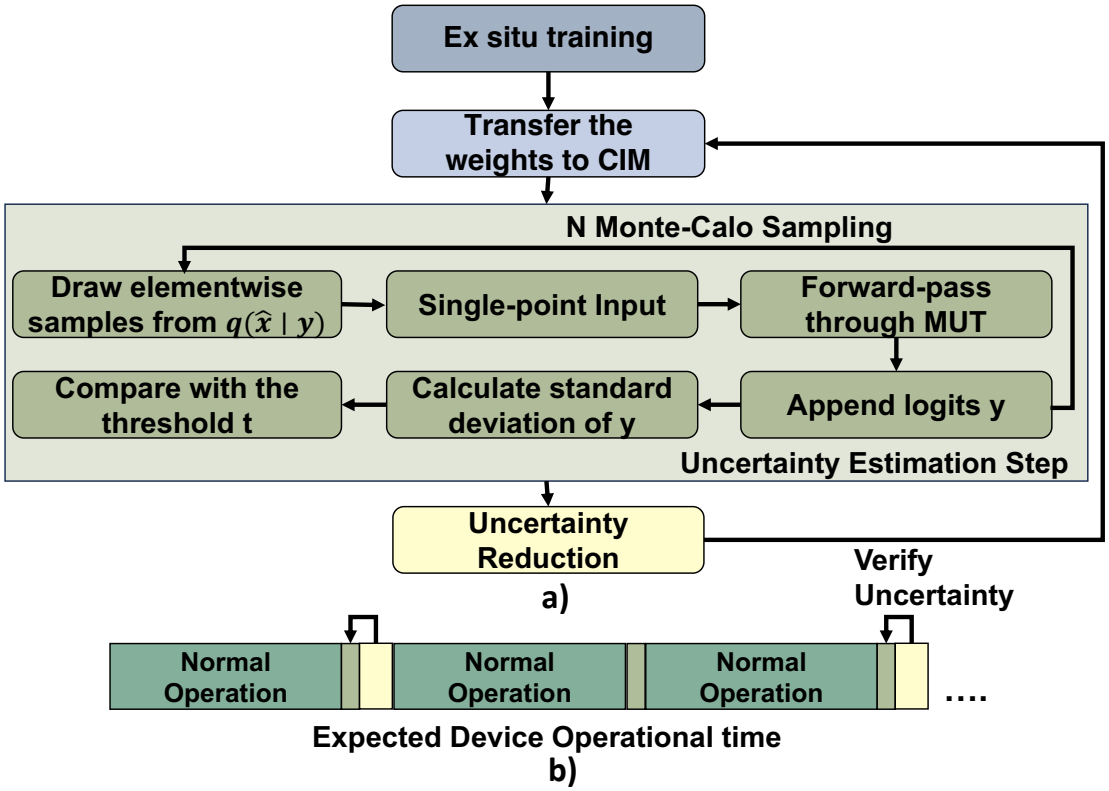


Figure 7.19.: Flowchart of the application of the proposed uncertainty estimation method during a) post-mapping but pre-deployment, b) post-deployment (online) operation.

The optimization process involves iterative adjustment of $\mu_{\hat{x}}$ and $\sigma_{\hat{x}}$ of $q(\hat{x}|y)$ elementwise to minimize the loss function. This can be achieved using a gradient-based optimization algorithm, such as stochastic gradient descent (SGD). At each step, we evaluate the loss and adjust the parameters of $q(\hat{x}|y)$ elementwise in the direction that reduces the loss.

The outcome of the proposed optimization process is a test vector $\hat{x}$ that, when used with a MUT, produces an output vector with minimized uncertainty σ_y, thereby achieving our goal of a distinguished output in the presence of functional uncertainty of the edge AI accelerator.

7.3.2.2. Application of Proposed Uncertainty Estimation Method

Our proposed method can estimate the functional uncertainty of the memristor-based edge AI accelerator at a given time during post-mapping, but before deployment of the NN and during online operation. If the model is uncertain, that is, the uncertainty estimates σ_y is more than a pre-defined threshold th, the uncertainty reduction method can be used. Common uncertainty reduction methods are re-training [151] and re-calibration [205]. After uncertainty reduction methods are applied, the functional uncertainty of the edge AI accelerator is re-estimated for verification. If the functional uncertainty is satisfactorily low, then the model resumes normal operation. Otherwise, a more sophisticated uncertainty reduction method or hardware replacement may be required before the model can resume normal operation.

The overall flow for the application of our memristive uncertainty estimation approach is depicted in Fig. 7.19. Latency for uncertainty estimation during pre-deployment may not be important as NN is not in operation. However, it is highly important during the online operation, as normal NN operation is paused while the uncertainty estimation process is carried out. Our uncertainty estimation approach is designed to keep this latency to a minimum. Consequently, our method improves the confidence and reliability of the prediction.

Table 7.14.: Summary of the evaluated models.

Model	Accuracy	# of Params.	Layers	Dataset	Input Shape
Classification (Supervised Learning)					
ResNet-20 [9]	92.60%	0.27×10^6	20	CIFAR-10 [223]	32×32
RepVGG-A0 [222]	94.39%	7.84×10^6	22		
ResNet-56 [9]	72.63%	0.86×10^6	56	CIFAR-100 [223]	32×32
MobileNet-V2 [219]	74.20%	2.35×10^6	53		
Inception-V3 [224]	77.29%	27.2×10^6	48	ImageNet-1k [8]	224×224
DenseNet-201 [218]	76.89%	20.0×10^6	201		
Semantic Segmentation (Supervised Learning)					
UNnet [164]	98.75%	7.76×10^6	23	Brain-MRI [216]	224×224
FCN [225]	91.40%	35.3×10^6	57	MS COCO [217]	224×224
Generative Method (Unsupervised Learning)					
DCGAN-G [226]	-	3.74×10^6	5	FASHIONGEN [227]	1×120
DCGAN-D [226]	-	2.93×10^6	5		64×64

7.3.2.3. Sampling From Bayesian Test Vector

We store element-wise $\mu_{\hat{x}}$ and $\sigma_{\hat{x}}$ of the Gaussian distribution (variational distribution) in the hardware. Element-wise sampling during the training and uncertainty estimation step is performed as follows:

$$\text{samples} = \mathcal{N}(0,1) \times \sqrt{\exp(\sigma_{\hat{x}})} + \mu_{\hat{x}}. \tag{7.22}$$

Where $\mathcal{N}(0,1)$ is a unit Gaussian distribution that can be implemented in software or hardware. The expression $\sqrt{\exp(\sigma_{\hat{x}})}$ calculates the standard deviation from the logarithm of the variance. It ensures numerical stability and avoids numerical underflow issues. The proposed element-wise sampling is inspired by the re-parameterization trick proposed by Kingma and Welling [203].

7.3.3. Evaluation

7.3.3.1. Simulation Setup

Evaluated Models Our method is evaluated on models from different deep learning paradigms, specifically classification, semantic segmentation, and generative methods with different state-of-the-art (SOTA) models [9, 222, 219, 224, 218, 164, 225, 226] with up to 201 layers. The number of classes in the benchmark datasets [223, 8] varies from 10 to 1000 for the classification tasks. The generator and discriminator models of the Convolutional Generative Adversarial Network (DCGAN) [226] are evaluated separately.

All pre-trained models were downloaded from the PyTorch Hub library and the GitHub repository [228]. Therefore, no changes to the training procedure are made. Our Bayesian test vector generation process did not alter the model parameters. Thus, the black-box nature of our method remains consistent. For the hyperparameter α, a value between $10^3 - 10^8$ is chosen.

Furthermore, we evaluated our approach on a range of input shapes, from 1×120 to 224×224. Table 7.14 summarized all models, their baseline accuracy, the number of parameters, layers, and the shape of the input image. All models in Table 7.14 are evaluated with a single MC sample and forward pass, $T = 1$. Therefore, the uncertainty estimation is done in a **single shot**.

Table 7.15.: Uncertainty estimation coverage for different NN models and datasets under varying noise strengths for both multiplicative and additive variations.

Model	Dataset	Multiplicative Variations					Additive Variations				
		η_0^1	η_0^2	η_0^3	η_0^4	η_0^5	η_0^1	η_0^2	η_0^3	η_0^4	η_0^5
Classification (Supervised Learning)											
ResNet-20	CIFAR-10	98.2%	100%	100%	100%	100%	98.1%	100%	100%	100%	100%
RepVGG-A0		99.7%	100%	100%	100%	100%	98.6%	100%	100%	100%	100%
ResNet-56	CIFAR-100	99.7%	100%	100%	100%	100%	97.5%	100%	100%	100%	100%
MobileNet-V2		99.7%	100%	100%	100%	100%	99.6%	100%	100%	100%	100%
InceptionV3	ImageNet-1k	98.2%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DenseNet-201		98.7%	100%	100%	100%	100%	99.4%	100%	100%	100%	100%
Semantic Segmentation (Supervised Learning)											
U-net	Brain-MRI	98.9%	100%	100%	100%	100%	99.2%	100%	100%	100%	100%
FCN (ResNet-101)	COCO	98.8%	100%	100%	100%	100%	99.7%	100%	100%	100%	100%
Generative Method (Unsupervised Learning)											
DCGAN-D	FASHIONGEN	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Fault and Variation-injection Framework Several studies have proposed a mathematical model of the non-idealities of the memristor. We use the variation model used in work [41, 128] that considers spatial and temporal variations and injects random additive and multiplicative Gaussian noise into the weights of pre-trained NNs. To control the severity of the variation, a noise scale η_0 is used. Similarly, we inject $\mathcal{P}_{flip}$% of bit- and level-flip-type faults into the weights of pre-trained NNs.

Uncertainty estimation coverage is calculated as:

$$\text{coverage} = \frac{\#\text{ of } \sigma_y \geq t}{\mathcal{R}} \times 100. \tag{7.23}$$

Essentially, it calculates the ratio between the number of times the uncertainty of the model σ_y is less than a predefined threshold th and the total fault runs $\mathcal{R}$. Also, each injected variation and fault is assumed to impact inference accuracy. We specifically choose noise scales $\eta_0^1 \cdots \eta_0^5$ and fault rates $\mathcal{P}_{flip}^1 \cdots \mathcal{P}_{flip}^5$ for each model that leads to a degradation of inference accuracy. Specifically, the values of η_0 and $\mathcal{P}_{flip}$ are chosen between $0.01 - 0.4$ and $0.02 - 1.8$%, respectively. However, subtle accuracy degradation is targeted because the uncertainty in these scenarios is much harder to detect. A Monte Carlo simulation is carried out for each scenario with $\mathcal{R} = 1000$ fault runs. For the th value, the uncertainty estimation value σ_y of the ideal NN is offset by a small constant $0 - 0.3$. Therefore, it is stored on the hardware alongside the test vector.

7.3.3.2. Estimating Uncertainty of Variations

Table 7.15 demonstrates the comprehensive evaluation of the coverage of our uncertainty estimation method across a spectrum of NN models under both multiplicative and additive variations with varying noise strengths, denoted by η_0. Remarkably, our method can consistently achieve 100% coverage or close to it under most of these diverse conditions. Furthermore, the high coverage percentage across diverse learning paradigms, specifically classification, semantic segmentation, and generative methods, emphasizes the applicability of our method in post-manufacturing and online operations of the CiM architecture. Therefore, reliability can be maintained under various noise conditions to improve confidence in the prediction.

The input (latent space) of GANs is Gaussian noise. It is equivalent to training an NN with Gaussian noise. As a consequence, the Generator of DCGAN is robust to both types of variation and has low uncertainty.

Table 7.16.: The evaluation of the proposed method in terms of coverage for estimating uncertainty due to both bit- and level-flip faults.

Model	Dataset	Bit-flip Faults					Level-flip Faults				
		$\mathcal{P}^1_{flip}$	$\mathcal{P}_{flip^2}$	$\mathcal{P}^3_{flip}$	$\mathcal{P}^4_{flip}$	$\mathcal{P}^5_{flip}$	$\mathcal{P}^1_{flip}$	$\mathcal{P}^2_{flip}$	$\mathcal{P}^3_{flip}$	$\mathcal{P}^4_{flip}$	$\mathcal{P}^5_{flip}$
Classification (Supervised Learning)											
ResNet-20	CIFAR-10	97.2%	99.0%	99.7%	100%	100%	98.7%	99.6%	99.8%	99.9%	100%
RepVGG-A0		98.9%	99.6%	100%	100%	99.9%	99.4%	99.8%	100%	100%	100%
ResNet-56	CIFAR-100	96.8%	99.4%	100%	100%	100%	98.7%	99.8%	100%	100%	100%
MobileNet-V2		99.2%	99.8%	100%	100%	100%	99.9%	100%	100%	100%	100%
InceptionV3	ImageNet-1k	99.6%	99.9%	100%	100%	100%	99.9%	99.9%	100%	100%	100%
DenseNet-201		99.5%	99.9%	100%	100%	100%	99.8%	100%	100%	100%	100%
Semantic Segmentation (Supervised Learning)											
U-net	Brain-MRI	98.3%	99.8%	100%	100%	100%	99.9%	99.9%	100%	100%	100%
FCN (ResNet-101)	COCO	99.0%	99.3%	100%	100%	100%	100%	99.4%	99.7%	100%	100%
Generative Method (Unsupervised Learning)											
DCGAN-Generator	FASHIONGEN	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
DCGAN-Discriminator		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Estimating the uncertainty of a variation robust NN model is unnecessary. Thus, we have not evaluated the uncertainty for this model.

7.3.3.3. Estimating Uncertainty of Bit- and Level-flip

Similar to uncertainty due to variations, our method can estimate uncertainty due to bit- and level-flip with consistently 100% coverage or close to it, as shown in Table 7.16.

Even though, the Generator of DCGAN is robust to variations, it is susceptible to bit- and level-flip-type faults. Therefore, we have evaluated its uncertainty.

7.3.3.4. True Positive Rates

We have shown that the out approach can achieve an ideal (or close to it) fault coverage based on injected faults. However, to further increase the confidence in the uncertainty estimates, we have modified equation 7.23 into:

$$\text{coverage} = \frac{\#\text{ of } (\sigma_y \geq th \; and \; A_{test} < A_{baseline})}{\mathcal{R}} \times 100. \tag{7.24}$$

This allows us to calculate true positive rates (TPR), which are *verifiable coverage* for uncertainty estimates. Here, A_{test} represents the inference accuracy after a fault or variation injection, and $A_{baseline}$ is the baseline inference accuracy. However, due to computational limitations, we have only evaluated this approach on a subset of the overall models shown in Table 7.14.

As depicted in Table 7.17, our uncertainty estimation approach can still achieve 100% coverage (most of the time) for uncertainty estimates in different fault rates and variations. This further underscores the robustness of our approach.

Table 7.17.: Evaluation of the uncertainty estimation coverage (true positive rates) with accuracy degradation verified in each step. The same noise scales η_0 and fault rates $\mathcal{P}_{flip}$ are used as in Tables 7.15 and 7.16.

Model	Dataset	Multiplicative Variations					Additive Variations				
		η_0^1	η_0^2	η_0^3	η_0^4	η_0^5	η_0^1	η_0^2	η_0^3	η_0^4	η_0^5
ResNet-20	CIFAR-10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
RepVGG-A0		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-56	CIFAR-100	99.1%	100%	100%	100%	100%	99.9%	100%	100%	100%	100%
MobileNet-V2		99.7%	100%	100%	100%	100%	99.1%	100%	100%	100%	100%
Model	Dataset	Bit-flip Faults					Level-flip Faults				
		$\mathcal{P}_{flip}^1$	$\mathcal{P}_{flip}^2$	$\mathcal{P}_{flip}^3$	$\mathcal{P}_{flip}^4$	$\mathcal{P}_{flip}^5$	$\mathcal{P}_{flip}^1$	$\mathcal{P}_{flip}^2$	$\mathcal{P}_{flip}^3$	$\mathcal{P}_{flip}^4$	$\mathcal{P}_{flip}^5$
ResNet-20	CIFAR-10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
RepVGG-A0		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-56	CIFAR-100	96.8%	99.4%	100%	100%	100%	99.4%	100%	100%	100%	100%
MobileNet-V2		98.2%	99.8%	100%	100%	100%	99.7%	100%	100%	100%	100%

Table 7.18.: Evaluation of the coverage of the proposed uncertainty estimate approach with faults and variations injected into a random subset (10-50%) of all layers. Here, the fault rate and the noise scale η_0 are kept constant.

Model	Dataset	Multiplicative Variations					Additive Variations				
		10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
ResNet-20	CIFAR-10	97.8%	100%	100%	100%	100%	88.8%	99.7%	100%	100%	100%
RepVGG-A0		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-56	CIFAR-100	96.9%	100%	100%	100%	100%	96.3%	99.8%	100%	100%	100%
MobileNet-V2		100%	100%	100%	100%	100%	99.7%	100%	100%	100%	100%
		Bit-flip Faults					Level-flip Faults				
ResNet-20	CIFAR-10	95.2%	99.7%	100%	100%	100%	96.2%	97.0%	96.7%	96.4%	97.6%
RepVGG-A0		97.3%	100%	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-56	CIFAR-100	98.0%	99.6%	100%	100%	100%	100%	100%	100%	100%	100%
MobileNet-V2		99.8%	100%	100%	100%	100%	100%	100%	100%	100%	100%

7.3.3.5. Layer-wise Uncertainty Estimation

We have extensively evaluated our approach when all the parameters of NN are affected by the memristive non-idealities. However, it is likely that not all the layers in an NN are affected by those non-idealities. Therefore, we have conducted further evaluations of our approach by randomly injecting faults and variations into $10 - 50\%$ of the layers of the CIFAR-10 and CIFAR-100 models. As demonstrated in Table 7.18, our method can achieve 100% uncertainty estimation coverage when $\geq 20\%$ of the layers of the NN are affected by variations or faults. Even when only 10% of the layers are affected, the uncertainty due to faults or variations that lead to $\geq 1 - 2\%$ accuracy degradation can be estimated with our approach.

7.3.3.6. Analysis of the Impact of Threshold Value on Coverage

Choosing the right value for the threshold th is important to achieve a high uncertainty estimation coverage. It implicitly reduces the risk of false positive or negative uncertainty estimates. We perform a series of analyses with different offsets for th. As demonstrated in Table 7.19, as the offset increases, the coverage gradually decreases to a close value 0%. Therefore, it is beneficial to use the threshold th as the baseline standard deviation of the ideal MUT. However, th should never be chosen below the SD of the ideal MUT. In this case, coverage could be high due to false-positive uncertainty estimation.

Table 7.19.: The effect of offset value of th on uncertainty estimation coverage. Evaluated on multiplicative variations with the same noise scale η_0 as Table 7.15.

Offset of Threshold th	Model RepVGG	Offset of Threshold th	Model ResNet-56	Offset of Threshold th	Model DenseNet-201
0.0	100%	0.0	100%	0.000	100%
0.40	100%	0.140	100%	0.040	100%
0.42	99.9%	0.150	99.7%	0.050	93.5%
0.43	97.0%	0.155	94.4%	0.051	77.9%
0.44	66.2%	0.160	74.0%	0.052	50.3%
0.45	18.6%	0.165	42.8%	0.053	22.2%
0.46	1.8%	0.170	17.1%	0.054	5.8%

Table 7.20.: Comparison of the proposed approach with the existing methods using different performance metrics. To ensure a fair comparison, the analysis of all approaches is conducted on the CIFAR-10 dataset. The memory consumptions for the test vectors and their labels are calculated based on the bit width reported by [37].

Methods	[37]	[117]	[118]	[41]	[229]	Proposed
Size of test vectors	10000	1024	10-50	16-64	$9^{V}/1^{Vi}$	**1**
# of test queries (normalized)	10000	1024	10-50	17	$9^{V}/1^{Vi}$	**1**
Memory overhead (MB)	0.015^{i}	234.42^{i}	0.154^{i}	0.0003^{i}	0.1105^{V} /0.0123^{Vi}	$\mathbf{0.02455}^{i}$
	245.775^{ii}	234.42^{ii}	0.154^{ii}	0.1966^{ii}	0.1105^{V} /0.0123^{Vi}	$\mathbf{0.02455}^{ii}$
Coverage (%)	99.27	98	76 / 84	100	$86.7^{V}/30^{Vi}$	**100**

[i] Re-training data is stored in hardware.
[ii] Re-training data is not stored in hardware.
[V] 32-bit floating-point, Re-training data is stored in hardware.
[Vi] 16-bit floating-point, Re-training data is not stored in hardware.

7.3.3.7. Resolution of Uncertainty Estimation

As mentioned previously, we have used a minuscule noise scale η_0 and P_{flip}, leading to negligible inference accuracy loss. Although our method can achieve 100% coverage for uncertainty estimates in those scenarios, it is important to find the boundary of coverage to determine the risk of false positive uncertainty estimates. Therefore, we have conducted several experiments on the CIFAR-10 and CIFAR-100 datasets, with even lower P_{flip} and noise scales. We have found that our method can estimate uncertainty with 100% coverage, even with $1-2\%$ accuracy degradation. However, when the accuracy degradation is very low, e.g., $\leq 0.5\%$, the number of false-negative uncertainty estimates can be as high as 5%.

7.3.3.8. Comparison With Related Works

We compare our approach to the related works with point estimate test vectors and Bayesian optimized test vectors, which employ a functional approach. Our single Bayesian test vector outperforms the methods proposed by Chen et al. [37], Li et al. [117], Luo et al. [118], and Ahmed et al. [41] on all metrics, as displayed in Table 7.20, even though we use only a single test vector, test query and forward pass. This implies that our method significantly reduces latency and energy consumption for uncertainty estimation compared to the other methods. The latency and the energy consumption are directly proportional to the number of test vectors used for estimating uncertainty. For instance, the testing method of Li et al. [117] requires 1024 test vectors, thereby necessitating 1024 times more matrix-vector multiplication operations and power consumption. In our comparisons, we assume that all the methods employ identical hardware implementation, NN topology, and memristor technology.

Moreover, our proposed approach consistently achieves 100% coverage in various fault and variation scenarios, exceeding the coverage rates of 30% to 99.27% achieved by the methods proposed by Chen et al. [37], Li et al. [117], Luo et al. [118], and A. Chaudhuri et al. [229].

In terms of the storage overhead of the Bayesian test vector, our method requires only 0.0245 MB to store the test vector, which is equivalent to storing only two (point estimate) test vectors. This is because our method requires the storage of mean and variance elementwise. Nevertheless, the memory requirement of our method is substantially lower than the other methods, regardless of whether re-training data is stored in hardware. Importantly, our method does not depend on storing re-training data in hardware to reduce memory consumption, unlike the methods proposed by Chen et al. [37] and Ahmed et al. [41].

In terms of sampling overhead for uncertainty estimation, there is a trade-off between the number of MC samples required T, the number of neurons in the penultimate layer of MUT C, and the storage overhead. For an MUT with a larger C, e.g., $C \geq 10$, only one sample is required, $T = 1$. In this instance, it is beneficial to take one MC sample before NN is deployed to CiM and store the sample on the hardware. Consequently, the number of MC samplings is reduced to one for the entire device operation, and the storage requirement for the Bayesian test vector is reduced to 0.0123 MB, a reduction of 2×. Overall, the overhead is the same as that of one test run for related methods. On the other hand, for an MUT with a small C, e.g., $C = 1$ *or* 2, sampling overhead can be reduced by taking multiple samples and storing them. In this case, the sampling overhead is reduced by $T\times$ in each of the following uncertainty estimation steps, but the storage overhead increases by $T\times$. For an edge device with limited memory, storing the parameters of a Bayesian test vector and taking samples in each uncertainty estimation step is more beneficial. In addition, the number of elements in an input determines how many E-samples are required in an uncertainty estimation step. Nevertheless, Elementwise sampling can be done in parallel since the input, e.g., an image, is a 3-D matrix.

7.3.4. Scientific Impact of This Work

The scientific impact of this work is summarized as follows:

1. **Bayesian Test Vector:** We introduce the concept of the Bayesian test vector, which has its element represented as a distribution rather than a single point. Therefore, it allows for more generalized low-cost testing that is scalable to any NN topology.
2. **Few Shot Testing:** We introduced a number of class-based NN tests. That is, a model with a smaller number of classes requires a few shots for testing, but a model with a large number of classes requires a single shot for testing. Therefore, our method is scalable to an NN in various classes.
3. **Gaussian Noise Injection for Fault-tolerance:** We show that when a model is trained with Gaussian noise injected into the input, it becomes variation tolerant. This concept can be further developed for variation tolerance in edge AI accelerators.

7.3.5. Section Conclusion

In this section, we propose a single Bayesian test vector generation framework to estimate the functional uncertainty of the memristor-based edge AI accelerator. The proposed Bayesian test vector is specifically optimized to provide low-uncertainty output for fault- and variation-free memristor-based edge AI accelerator. Our method requires only single (element-wise) sampling from the distribution of the Bayesian test vector and a single forward pass on a larger model. Thus, the overhead associated with our approach is minimal. In addition, we proposed an application of our uncertainty estimation approach in the pre-deployment and post-deployment scenarios of an NN to MHA. We have consistently demonstrated high uncertainty estimation coverage of our approach on various NN topologies, tasks, fault rates, and noise scales related to variations. Our work improves confidence in the predictions made by memristor-based edge AI accelerator.

8. Explicit Testing of Bayesian NNs

BayNNs offer substantial benefits over conventional NNs and can inherently capture and estimate the *uncertainty* of their predictions. As mentioned before, Spintronics-based CiM architectures are a promising solution for the hardware realization of BayNNs as they mitigate some of the inherent computational costs, balancing high-performance demands with the constraints of resource-limited devices. However, Spintronics-based CiM-implemented BayNN suffers from various non-idealities that impact their accuracy and violate functional safety. Addressing these reliability issues with proper periodic in-field testing is critical to ensuring the reliability of BayNNs in real-world safety-critical applications.

However, testing Dropout-based BayNN in Spintronics-CiM presents unique challenges, primarily due to the stochastic nature of its output. This section aims to address this and is based on our IEEE ETS24 [190] paper.

8.1. Methodology

8.1.1. Problem Statement

Dropout-based BayNNs and their Spintronics implementation represent a distribution of output over possible models, rather than a single model. Thus, the Spintronics-CiM outputs are different for the same input, as shown in Fig. 8.1 (a). We refer to this characteristic as the repeatability or high-variance problem. Consequently, functionally testing BayNNs in Spintronics-CiM introduces several unique challenges due to the non-deterministic nature of their output due to the application Dropout. Due to the repeatability problem, traditional functional testing-based approaches for testing Spintronics-CiM can lead to too many false positives or low true positive rates. This is because traditional approaches compare the expected output, label, or distribution of Spintronics-CiM to a pre-defined deterministic one without considering the stochasticity of BayNNs. Therefore, our previous works [41, 221, 42] targeted for testing conventional NN is not applicable here, and a novel approach should be devised. Also, unlike the ensemble-based approach in [230], the uncertainty estimates of BayNN in Spintronics-CiM are also stochastic, even for the same input, as shown in Fig. 8.1 (b). This creates an additional challenge in testing BayNNs in Spintronics-CiM based on their uncertainty.

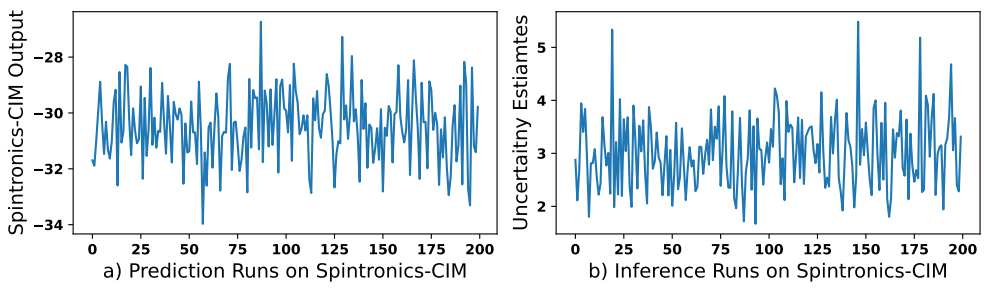


Figure 8.1.: Stochasticity of a) Spintronics-CiM output (logits values) and b) uncertainty estimates for the same input for 200 different predictions and inference runs, respectively.

Additionally, the input space of BayNNs can be very large, especially for models with many hidden layers. This is because Dropout-based BayNNs in Spintronics-CiM effectively create a separate sparse model for each forward pass. Therefore, there are many possible inputs that the BayNN in Spintronics-CiM could be tested with. Exhaustively testing the model on all possible inputs would be computationally infeasible.

Furthermore, the test cost of BayNN in terms of latency and power presents a significant challenge for resource-constrained devices or real-time applications. In these applications, the availability of the device is an important factor and cannot be unavailable for too long while a test operation is performed. However, Bayesian inference requires numerous forward passes to obtain a prediction for the input sample. Therefore, the number of test vectors should be minimal.

8.1.2. Automatic Test Generation Framework

To address the challenges mentioned before, we propose a novel *sample-based automatic test vector generation framework* to test BayNNs in Spintronics-CiM. In our approach, a small subset of training data is sampled based on their variance in the output. Here, variance is treated as a measure of the repeatability of the predictions. We hypothesized and observed that, despite the stochastic output of the model, the variance of inputs varies from one to another. Inputs with low variance are close to the deterministic model and are thus more suitable for testing BayNN in Spintronics-CiM, as they yield more interpretable outputs and uncertainties.

Our approach involves performing statistically significant repetitive inference runs for each training data sample. Subsequently, the variance in the uncertainty is calculated for each input. Additionally, training data is ranked based on their variance, with lower-variance samples receiving higher priority for selection. Lastly, several lower-variance training data points are stored in the hardware as test vectors.

In our approach, test vectors are sampled from the training dataset because we observed that the uncertainty of the distribution of training and validation data overlaps when no random data augmentations are applied. Therefore, it can be stated that the uncertainty of the prediction is the same regardless of the input data seen during the training, as long as they are of the same distribution. Therefore, no holdout data are required, which could otherwise reduce the available data for model training or validation. Consequently, our approach is particularly advantageous in scenarios with limited data.

8.1.3. Proposed Fault Detection Approach

For fault detection, we hypothesize that as the Spintronics device deviates from its initial state or is faulty, the uncertainty of the prediction increases. Our hypothesis is grounded in the fact that as the distribution of input to the Spintronics-CiM shifts away from the training distribution, e.g., due to random noise introduced by the sensor or by dynamic environmental conditions such as rain, snow, or fog, the uncertainty of prediction increases as demonstrated in several existing works [231, 49, 86, 85]. In scenarios where faults and variations occur on the Spintronics cells storing BayNN weights and buffer memories storing activations of hidden layers, inputs to subsequent layers will also change from their expected values, effectively creating *intermediate out-of-distribution* data, even though initial input to the Spintronics-CiM remains within the in-distribution range. Therefore, the uncertainty of the prediction is expected to increase.

Our objective is to detect these changes in prediction uncertainty as a means of testing the BayNN implemented in Spintronics-CiM. We define a predefined range for the uncertainty of the BayNN given the test vectors. The BayNN model is classified as faulty if the uncertainty distribution changes from the predefined distribution. Otherwise, it is not faulty.

We fit the uncertainty distribution of the test vectors to a Gaussian distribution, as they are mathematically well-defined and easy to work with. Afterward, we evaluate the mean μ and standard deviation σ of the

uncertainty distribution to estimate the baseline (fault-free) range of the prediction uncertainty. Specifically, we define the two bounds, b_1 and b_1, representing the upper and lower bounds of the distribution based on the empirical rule of probability. The rule states that 99.7% of the values of a Gaussian distribution are within three standard deviations of the mean. Therefore, $\mathcal{B}_1$ and b_1 are defined as $\mu + 3 \times \sigma$ and $\mu - 3 \times \sigma$, respectively.

Lastly, during each online testing phase of Spintronics-CiM, the test vectors are applied sequentially to the model as *test queries*. If the uncertainty of the prediction is above or below the boundaries, the model is classified as faulty. Therefore, in our approach, only lightweight checks are required in the model output. Afterward, thorough testing is required, for example, to localize the faults and perform re-training [151] or re-calibration [205, 204] to mitigate the impact of faults and variations.

8.1.4. Reduction of False Positives Rate

A theoretically sound approach to address the repeatability problem and reduce the false-positive rate would be to test BayNNs based on expected output. Specifically, predictions and uncertainties are derived from the mean of multiple inference results, with each inference result obtained after multiple forward passes, as described earlier. However, this approach is impractical. For example, if the expected uncertainty is determined after 10 inferences, each requiring 20 forward passes, the total computation for a single test vector reaches 200 forward passes, which is prohibitively expensive.

Therefore, we propose a low-cost *vote-based approach*. Specifically, unlike in work [41], multiple test queries contribute to the fault identification process. However, to minimize test costs in terms of latency and power, we limit the length of the query sequence to the minimum when a close-to-ideal (less than 10%) false positive rate is achieved.

8.2. Evaluation

8.2.1. Simulation Setup

The proposed method is evaluated across three state-of-the-art Spintronics-CiM-implemented dropout-based BayNN methods: SpinDrop [85], SpatialSpinDrop [86], and ScaleDrop [48]. All methods are implemented with the widely used ResNet-18 topology and the CIFAR-10 benchmark dataset.

The test vector is generated after performing 200 repetitions of inference, and with each inference, 20 Monte Carlo samples of the Dropout mask were performed. For fault detection, a positive test query length of four was used, and 100 test vectors were used. We have performed the Monte Carlo fault simulation with a 1000 fault injection for each fault or variation rate. In total, we have performed 60, 000 fault injection campaigns at different locations of Spintronics-CiM as mentioned in Section 2.10.

The proposed method evaluates fault coverage, which states how many injected faults are detected. Faults are treated as benign if the accuracy does not degrade noticeably. Otherwise, they are critical faults. Fault coverage for critical faults represents true positive rates (TPR), and without faults represents false positive rates (TPR).

8.2.2. Fault Models For Spintronics-CiM-based BayNN

Permanent faults in Spintronics cells and buffer memory are modeled as stuck-at faults, as is usually done, with logic values always fixed at ’0’ or ’1’ [232]. In binary BayNNs, these translate to weights and activations being persistently stuck at ’-1’ or ’1’. In the Dropout module, the permanent fault results in a constant Dropout mask of ’0’ or ’1’. Thus, a word-line in the crossbar can always be inactive or active. Similarly, the widely used bit-flip fault model is used for transient faults. In the bit-flip fault model, the

logic value in the Spintronics cells and the buffer memory randomly shift from '0' to '1' or vice versa. For binary BayNNs, this means random flipping of logic values between '-1' and '1'. In the Dropout module, this manifests itself as random bit flips in the Dropout mask. Thus, a word-line that was originally inactive can be active, and vice versa.

Furthermore, existing work shows that conductance variations can be modeled as additive and multiplicative Gaussian variations [75] and variations in the resistive states of MRAM devices lead to variations in the current sum [67]. Thus, based on these works, conductance variations in Spintronics devices are modeled as additive and multiplicative Gaussian variations in the MAC result.

Lastly, due to fluctuations in the switching current of the Dropout module, the Dropout probability can vary. Based on works [85], variation in the Dropout probability is represented with a Gaussian distribution, the mean representing the original Dropout probability.

8.2.3. Fault Sensitivity Analysis of BayNN on Spintronics-CiM

BayNNs implemented in Spintronics-CiM can withstand up to 5% of stuck-at and bit-flip faults in MTJs and buffer memories, as shown in Fig. 8.2. In particular, BayNNs are more fault-tolerant in the case of stuck-at faults compared to bit-flip faults. Specifically, they can withstand up to 10% of stuck-at-faults in buffer memories that store binary activations. In terms of BayNN methods, scale dropout-based BayNNs are more fault-tolerant and can tolerate up to 15% of stuck-at-faults in buffer memories. In contrast, the inference accuracy gradually decreases with the additive and multiplicative types of MTJ conductance variations, as shown in Fig. 8.3. Nevertheless, all BayNN methods show an overall trend of a reduction in accuracy as the fault rate increases. Therefore, this emphasizes the necessity for fault detection to ensure the functional safety and reliability of BayNNs.

8.2.4. Analysis of Fault Coverage

As shown in Figs.8.4 (a) and (b), our proposed approach can predominantly achieve 100% fault coverage for critical faults on MTJs and buffer memories of Spintronics-CiM. Similarly, our proposed approach can consistently achieve 100% fault coverage for conductance variations of Spintronics devices, as depicted in Fig.8.4 (c). In the worst case, our approach can achieve a fault coverage of 89.10%. Consequently, our approach is more effective in detecting faults in the buffer memories and conductance variations. Specifically, even when the accuracy drop is marginal, i.e., 0.30%, our proposed approach can achieve 100% fault coverage. In terms of BayNN methods, our approach is particularly effective with the SpinDrop and SpatialSpinDrop methods, which are more fault-sensitive compared to the ScaleDrop method. Note that the fault coverages mentioned here are TPR and our approach can consistently achieve an ideal 100% TPR value.

Furthermore, our approach can also achieve up to 100% fault coverage for benign faults. Detecting benign faults before they catastrophically degrade accuracy is beneficial, especially for faults, such as retention faults, that accumulate over time.

8.2.5. Analysis of False Positive Rate (FPR)

Although achieving high fault coverage is important in the case of critical faults and conductance variations, achieving a low FPR is equally important in reducing false alarms. As shown in Fig. 8.5, our approach gradually reduces the FPR to an ideal value of 0% with a positive test query length of 5. As mentioned in Sec. 8.2.1, evaluation of test coverage performed with a positive test query length of 4, which results in an acceptable FPR of 10% and less.

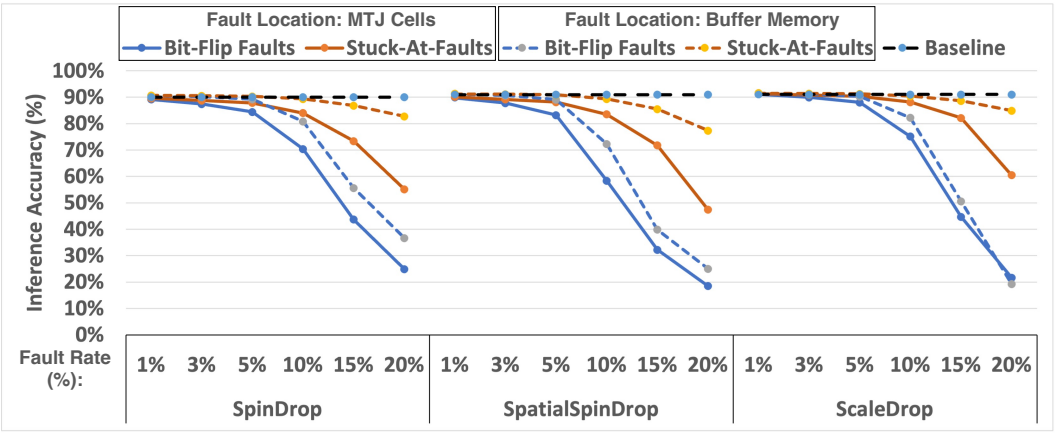


Figure 8.2.: Impact of Inference accuracy of BayNNs implemented in Spintronics-CiM with different bit-flip and stuck-at-fault rates, compared to a baseline without faults. **It is recommended to view this figure in color.**

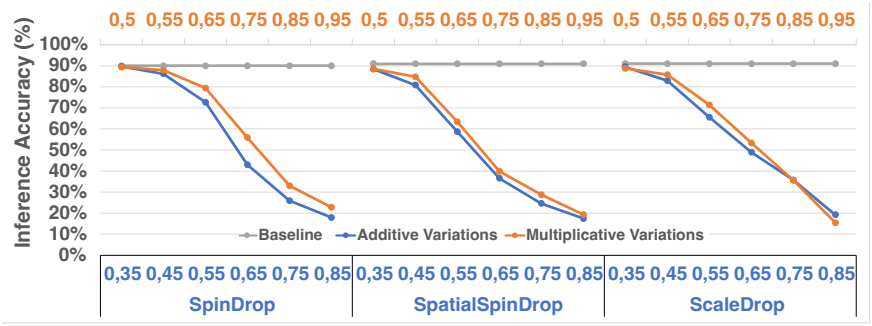


Figure 8.3.: Comparison of the impact of inference accuracy of BayNNs implemented in Spintronics-CiM with conductance variations relative to a fault-free baseline. **It is recommended to view this figure in color.**

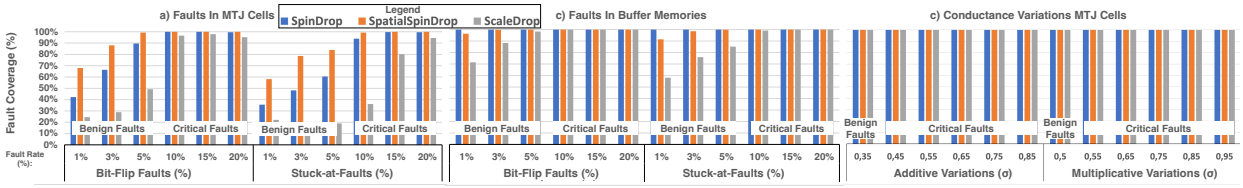


Figure 8.4.: Fault coverage of proposed approach on various Spintronics implemented BayNN methods under varying bit-flip and stuck-at faults rate a) affecting Spintronics cells that store weights, b) buffer memories that store intermediate activation, and c) different conductance variations in Spintronics. **It is recommended to view this figure in color.**

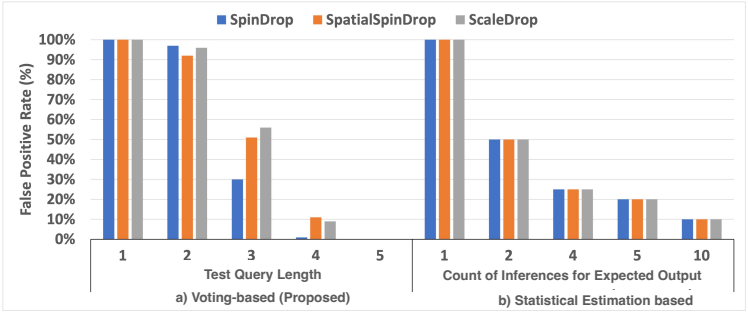


Figure 8.5.: False positive rate (lower the better) of a) proposed voting-based approach, and b) theoretically grounded estimation-based approach. **It is recommended to view this figure in color.**

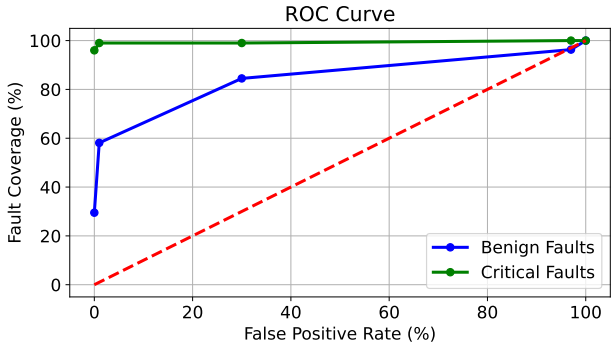


Figure 8.6.: ROC curves for benign and critical faults with varying positive test query lengths. **It is recommended to view this figure in color.**

There is a trade-off between fault coverage and FPR given by the positive test query length, as shown in the Receiver Operating Characteristic (ROC) curve in Fig. 8.6 for the SpinDrop method. In an ROC curve, the closer a curve is to the top-left corner, the better the performance. With our proposed approach, both the curves for benign and critical faults perform well because they are above the random diagonal line. Specifically, the curve for critical faults does not change with FPR. This suggests that our approach can achieve 100% fault coverage even with a positive test query length of 1 and is better at detecting critical faults than benign ones. In contrast, the coverage of the benign fault is highly dependent on the positive test query length. In practice, we suggest using a positive test query length of 4 as it can detect a sufficiently large number of benign faults and the majority of critical faults at low FPR.

8.2.6. Analysis of Non-ideal Dropout Module

We found that BayNNs [85, 48, 86] are robust to faults (stuck-at and bit-flip) and variation in the switching current of the Dropout modules and do not impact the inference accuracy with up to a 20% fault rate. Similarly, uncertainty estimates of BayNNs are generally not affected. However, in the SpinDrop and ScaleDrop methods, bit-flip and stuck-at-0 faults lower the uncertainty, that is, making the BayNN *overconfident*. In these scenarios, the faults are considered critical, and our approach can achieve 100% coverage. Furthermore, if the Dropout modules are shared within a layer and all the other layers, then a fault in the Dropout module affects all the neurons of a layer and other layers. In this case, stuck-at faults make all the word-lines of Spintronics-CiM always inactive or active and make the uncertainty estimates *zero*, i.e., the same as conventional NNs. In this case, our approach can also achieve 100% coverage.

8.2.7. Overhead Analysis and Comparison to Related Works

Regarding memory overhead, our approach requires 0.31 MB of memory to store test vectors. Furthermore, for a single test query, our approach consumes energy of 12.09, 3.88, and 1.15 μJ for the SpinDrop, Spatial-SpinDrop, and ScaleDrop methods on ResNet-18, respectively. Here, the energy consumption depends on the specific implementation of BayNN. For a BayNN status check, the overall energy consumption required depends on the number of test queries. In the worst case, our method requires 1209, 388, and 115 μJ energy, respectively, for the 100 test queries. In comparison, the statistical estimation-based approach requires 10× energy consumption for a test query to achieve 10% FPR.

To the best of our knowledge, this paper proposes the first work on testing BayNN on Spintronics-CiM. The other related work [230], although not directly comparable, estimates the uncertainty of the model due to process variations using an ensemble-based approach. They do not report fault coverage, but an 80.4% reduction in calibration error (a measure of uncertainty) is reported. Also, their work has a memory overhead of 0.45 MB and an energy consumption of 6.02 μJ per test query. This results in 602 μJ for the 100 test vectors in an explicit testing scenario.

8.3. Scientific Impact of This Work

The scientific impact of this work can be summarized in the following:

1. **Testing Stochastic NNs in Edge AI**: We propose a comprehensive testing method that addresses the unique challenges posed by the stochastic nature of BayNN outputs, which traditional testing methods cannot efficiently handle.
2. **ATPG for BayNN**: We proposed a repeatability ranking-based automatic test pattern generation method that achieves high fault coverage while using only 0.2% of training data as test vectors.
3. **Scalability**: Our testing methodology is highly scalable and can be applied to other memristor technology-based CiM, other edge AI accelerator architectures, neural network topologies, datasets, and tasks. The relatively low testing cost makes it particularly suitable for real-time application, ensuring broad applicability and versatility.
4. **Monitoring Uncertainty Estimates as a Means to Detect Fault**: Many of the existing BayNN show high uncertainty due to OOD data. In this work, we show that uncertainty distribution also changes due to faults or variations. We utilized this aspect to detect faults and variations in BayNN effectively.

8.4. Section Summary

In this section, we propose for the first time a test generation and online testing framework for Dropout-based BayNN implemented in Spintronics-CiM. We also present fault analysis of different non-idealities of Spintronics-CiM. Our approach can consistently detect 100% of the critical faults at different locations. Furthermore, our approach requires only 0.2% training data as test vectors and a simple check at the BayNN output.

9. Concurrent Testing NNs

In this section, we focus on the challenge of concurrently self-testing BNNs. Due to their lower bit-precision and computation cost, they are particularly suited for resource-constrained edge AI accelerators. Concurrent testing is particularly suitable for always-on safety-critical applications that do not tolerate any system downtime and expect continuous operation. However, keeping a low false positive rate and high fault coverage is challenging. We address these challenges in this section. This section is based on our paper [233].

9.1. Problem Statement

Concurrent self-testing of BNNs is a challenging task. As demonstrated in Figure 9.1, the learned representations of the backbone network of a ResNet-18 BNN exhibit minimal changes and overlap even at a 20% soft fault rate. Afterward, these learned representations are received by the classification head. Therefore, the output distribution can change marginally from the fault-free one.

The primary objective is to devise a concurrent self-testing method that meets the following criteria: **a)** Requires no additional forward pass through the network. **b)** Achieves a low False Positive Rate (FPR), minimizing the number of false alarms. **c)** Attains a high True Positive Rate (TPR), ensuring that faults in CiM architectures are accurately identified, **d)** is suitable for BNNs, and requires lightweight checks online for testing.

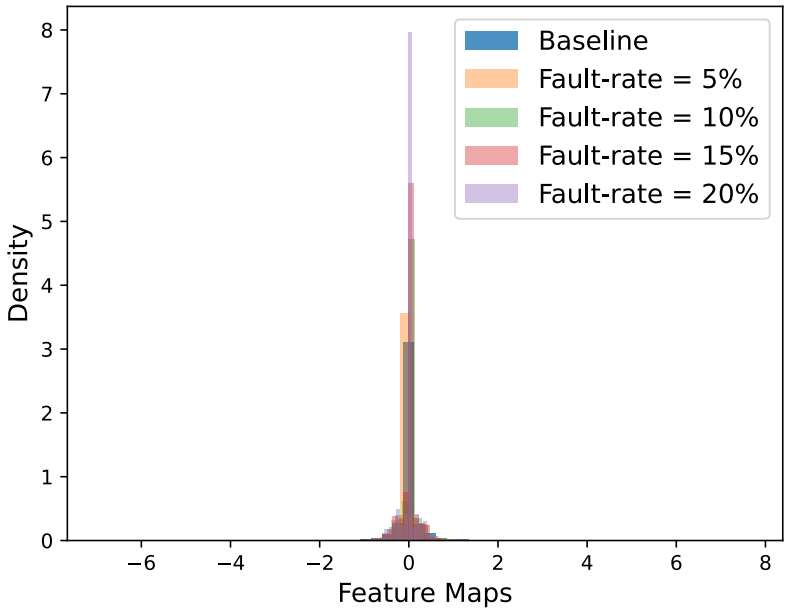


Figure 9.1.: Change in the distribution of the feature maps due to soft-faults modeled as bit-flips of memory cells that store weights (see 9.3.1) on binary ResNet-18 trained on CIFAR-10. **It is recommended to view this figure in color.**

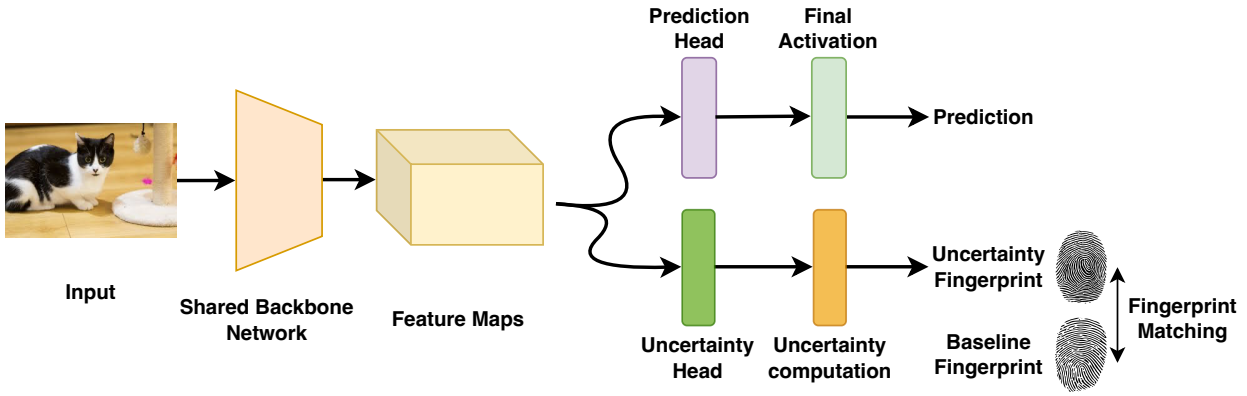


Figure 9.2.: Two-Headed model with point estimate parameters for concurrent self-testing and uncertainty estimation. The model is generalizable with existing NN topologies.

9.2. Methodology

9.2.1. Uncertainty Fingerprint

In this chapter, we introduce the *uncertainty fingerprint* ($\mathcal{U}$) concept, a specialized metric for concurrent fault detection in CiM architectures, especially in always-on safety-critical applications. The uncertainty fingerprint is specifically designed to be the output of a dedicated head in a dual-head NN topology as a form of alternative prediction. We refer to that head as the "uncertainty head." Fig. 9.2 shows the block diagram of the proposed topology.

In this work, the uncertainty fingerprint is defined as the maximum value (`max()`) produced by the uncertainty head. We proposed a tailored learning objective for this outcome. During the training phase, the uncertainty head is explicitly tuned with the proposed objective of bringing the maximum output value to *one* for each input during inference. However, the exact value of the uncertainty fingerprint, even in the fault-free state of the CiM architecture, can vary from one input to another and is itself a distribution. Consequently, the optimization objective ensures that the uncertainty fingerprint distribution is centered around one. The main goal of the optimization is to establish a "signature" or "fingerprint" based on the CiM architecture's fault-free state. Note that optimization does not require any faulty behavior of CiM architectures or explicit fault injection.

We hypothesize that as the memory cells or buffer memory of CiM architectures change due to permanent or soft faults, the distribution of uncertainty fingerprints can change, i.e., the distribution shifts to the left or right, thus, making it distinguishable from the pre-defined fault-free distribution. This is because the output of the uncertainty head is a linear transformation of its input, which is the output of the backbone network. To reiterate, faults in the memory elements and intermediate results of a layer would lead to the input of the following layer being faulty. The cascading effect of this will propagate the output backbone network and, ultimately, the CiM architecture. That means that both the prediction and the uncertainty head will receive input that is different from the fault-free one. Consequently, the output of the prediction head will be incorrect, and the uncertainty head will be different from its baseline.

Therefore, by matching the expected uncertainty fingerprint value online in real-time for *each prediction*, we propose to concurrently self-test the CiM architecture. If the uncertainty fingerprint of the CiM architecture matches, then a prediction is classified as fault-free, otherwise, it is classified as faulty.

9.2.2. Dual-Head Model

To reiterate, in modern NN topologies design, task-specific heads are increasingly used [43]. To obtain the proposed uncertainty fingerprint of the model in a single shot without reducing the accuracy, we introduce

an additional head to the NN, *uncertainty head*. The uncertainty head is typically a linear layer with a predefined number of neurons that is independent of the number of classes in the dataset. The specific number of neurons is a hyperparameter that should be optimized to improve fault detection accuracy. Unlike conventional NN topologies, our proposed topology can be self-tested. Both the uncertainty and the prediction heads share the same (unchanged) backbone network.

If the prediction head is also a linear layer, the input that both the uncertainty and prediction heads receive can be the same. Otherwise, an additional pre-processing layer, such as adaptive average pooling, can be applied before the uncertainty head. The adaptive average pooling layer reduces the spatial dimensions of the feature maps to a single value. Thus, it significantly reduces the size of the weight matrix of the uncertainty head. Hence, our proposed approach can potentially be applied to various NN topologies, such as fully convolutional NN (FCN), in which the prediction head is a convolutional layer.

9.2.3. Training Objective

We propose a two-step training approach for our objective. In the first step, the NN is trained using the gradient descent algorithm, minimizing task-specific loss, such as cross-entropy. In this step, only the output of the prediction head is taken into account.

In the second step, we *freeze* the rest of the model, including the prediction head, and train only the uncertainty head. The term "freeze" here means that the gradient is not calculated with respect to the loss value, and the associated parameters and variables will not be updated.

We propose a fingerprint loss function for this step of the training. It is defined as:

$$\mathcal{L} = \alpha \times \frac{1}{B} \sum_{b=1}^{B} (1 - max(\mathcal{F}_{\theta'}(\mathbf{x}_b)))^2 \tag{9.1}$$

Here, α is a hyperparameter that controls the strength of the loss function, $\mathbf{x}$ is the NN input, $\mathcal{F}(.)$ denotes the NN with θ' summarizing all the parameters of the NN excluding the parameters of the prediction head, and B is the batch size. The loss function encourages the uncertainty fingerprint for each input to be close to one. Consequently, it encourages the distribution of uncertainty fingerprints to be centered around one. It can be considered similar to the conventional mean squared error (MSE), but is applied to the uncertainty head.

For training and baseline uncertainty fingerprint estimation, we divide both the training and validation datasets with an 80:20 split, where 80% of the training data trains the functional task and 20%(fingerprint data) trains the uncertainty head to encourage its maximum output close to one. Since the NN was not trained on fingerprint data, the inputs to the uncertainty head from the backbone network resemble those during inference. Consequently, this enables the uncertainty head to learn outputs akin to those expected during inference, aligning its learned representations closer to the inference scenario. Note that random data augmentations and stochastic regularization approaches, such as Dropout, should be avoided during uncertainty head training. Otherwise, the uncertainty head might learn to be robust to input variations, leading to a lower fault detection rate.

9.2.4. Online Concurrent Self-test

We propose boundary-based online testing, which is a lightweight and effective way to detect faults in an online, resource-constrained edge devices. We pre-compute the boundary values, l and h, offline. This range suggests that most of the uncertainty fingerprints of the fault-free model fall within this range. Then, during the online operation, if an uncertainty fingerprint is observed outside this range, it suggests that

the data on the memory cells storing weights or buffer memory storing intermediate activations of CiM architectures have changed due to fault occurrence.

The main idea behind this approach is to establish a "normal operating range" for the CiM architecture's uncertainty fingerprint. To do this, we compute the $l = 2.5\%$ and $h = 9.5\%$ quantiles of the uncertainty fingerprint for the fault-free model. During the online operation, two scenarios can arise regarding the distribution of uncertainty fingerprint score. If the score is less than l, it may indicate a leftward drift in the distribution. In contrast, a score greater than h could suggest a rightward drift. In both cases, these shifts signal that the uncertainty fingerprint score is different from what was observed in the fault-free model and the faults in the CiM architecture. Therefore, a fault is detected if it satisfies this condition:

$$\text{Status of the model} = \begin{cases} \text{Faulty} & \text{if } \mathcal{U} < l \text{ or } \mathcal{U} > h, \\ \text{Fault-Free} & \text{otherwise.} \end{cases}$$

As a result, our approach requires only lightweight checks to detect faults. The < (less than) and > (greater than) operations can be implemented in software or even hardware using comparators available in existing hardware accelerators.

Our approach, similar to the work of Gavarini et al. [119], may result in false positives or negatives, which need to be minimized. However, unlike [119], we propose two strategies to reduce them. The number of false positives or negatives is influenced by the boundaries of the uncertainty fingerprint. Therefore, it is crucial to tune the boundaries for each dataset or task to maximize coverage and minimize false positives.

Initially, we obtain the boundary values from the fault-free model using a dataset that closely represents the expected real-world data distribution that the model will encounter during inference. Thus, the quantiles are determined on 20% of the validation data, providing an unbiased estimate of the uncertainty fingerprint distribution. The remaining 80% of the validation dataset is used for evaluating performance and fault coverage.

Since we choose boundaries as 2.5% and 95% quantile values, it effectively ignores the tails of the uncertainty fingerprint distribution, treating them as anomalies. In some datasets and models, these anomalies can skew the values of l and h. Therefore, we compute the Z-score of the uncertainty fingerprint distribution and adjust l and h on data with a Z-score of less than two. Z-scores help identify anomalies in a distribution, with scores above two indicating an anomaly.

Note that the boundaries can be adjusted online if necessary, especially when false positives are encountered. The proposed concurrent detection method serves as an initial line of defense. It can trigger more rigorous (explicit) tests, e.g., to find the location of faults, followed by appropriate mitigation approaches, such as retraining [151].

9.3. Results

9.3.1. Simulation Setup

Fault Modeling and Injection We model permanent faults using the widely used "Stuck-at" fault model, where the memory cells appear to be held exclusively high or low. This translates to stuck-at-one or stuck-at-zero in memory cells and buffer memory of CiM architectures that store weights and activation, respectively. In the case of multi-bit (K-bit) weights and activation, faults can cause stored weights and activation to be stuck in any of the 2^k states. On the other hand, soft faults are modeled as "random bit flips", implying that the memory element contains random but inaccurate values. This translates into memory cells in high states randomly switching to low states, and vice versa. At the algorithmic level, this

Table 9.1.: Comparison of the proposed method with the baseline method with different topologies.

Method	Topology			
	ResNet-18	PreActResNet	VGG-9	SegNet
Baseline	90.60	95.40	82.98	93.65
Proposed	90.68	95.54	83.0	93.53

means that the BNN weights and activations randomly switch to −1 from +1, and vice versa. However, for multi-bit weights and activations, this means that they can flip to any of the $2^k - 1$ other states.

Furthermore, We perform the Monte Carlo fault simulation with 100 fault runs. Monte Carlo fault simulation is widely used to evaluate NN performance. Faults are injected into memory cells that store weights and activations of pre-trained NN at random locations given by Bernoulli's distribution. Specifically, bit-flip faults are injected into the NN activation (after the sensing circuit digitizes the weighted sum) during each forward pass to simulate faults that occur while CiM architectures compute inference results.

Each layer of the NN is mapped to several CiM memory arrays of dimension 64×64 and their weighted sum is accumulated to obtain the final MAC results. We have abstracted circuit-level details of emerging memory technology and performed the evaluation of our approach in the widely used PyTorch framework.

Evaluated Models and Dataset The proposed method is evaluated across four state-of-the-art CNN topologies: ResNet-18, PreActResNet-34, VGG-9, and SegNet (a FCN topology), trained in the CIFAR-10, SVHN, Flowers-102 (classification tasks), and breast cancer segmentation (biomedical semantic segmentation) datasets, respectively. These topologies diverge not only in architectural depth, spanning 11 to 34 layers, but also in the number of target classes, ranging from 2 to 102. All models have binary (+1 and −1) weights and activations using the IRNet algorithm [47], but the bit width of the SegNet activation is increased to 4-bit as the task is much harder.

Evaluation Metrics In terms of evaluation metrics, we report fault coverage, which is defined as the percentage of validation data flagged as "faulty". In a fault-free model, fault coverage represents FPR, and in a faulty model, it represents TRP. In the case of FPR, the lower the better, but for TPR, the higher the better. In the ideal case, FPR is 0% and TPR is 100% are desired at the same time, as they would result in fewer false alarms and prevent potentially incorrect predictions from reaching users.

9.3.2. Inference Accuracy

The proposed dual-head model maintains comparable performance across various NN topologies, as illustrated in Table 9.1. This shows that the proposed approach does not have an impact on inference accuracy. In particular, both the proposed and baseline models were trained using 80% of the training data (see Section 9.2.3). A slight improvement in accuracy, estimated between 0.1 and 1%, could be observed if the baseline NNs were trained with 100% of the training dataset.

9.3.3. Analysis of Permanent and Soft Faults Coverage

Figs. 9.3(a) and (b) show the effect of permanent and soft faults at different rates (occurring in the memory cells of the CiM architecture that stores weights) on the inference accuracy in different datasets. It can be observed that as the fault rate increases, the inference accuracy decreases across all datasets. However, the effect of faults can be considered on a case-by-case basis.

The inference accuracy of the classification models shows a noticeable decline beyond the fault rate of 5% and 10%, for permanent and soft faults, respectively. However, for the Flower-102 model, it shows greater

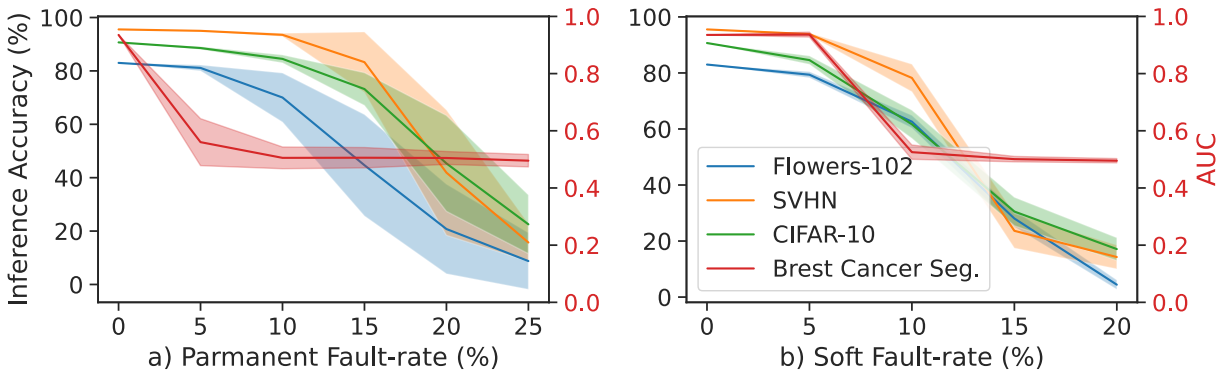


Figure 9.3.: Impact of inference accuracy due to (a) permanent faults and (b) soft faults impacting memory cells of CiM architectures that store weights. Shaded regions indicate the one standard deviation variation around the mean inference accuracy or AUC scores. **It is recommended to view this figure in color.**

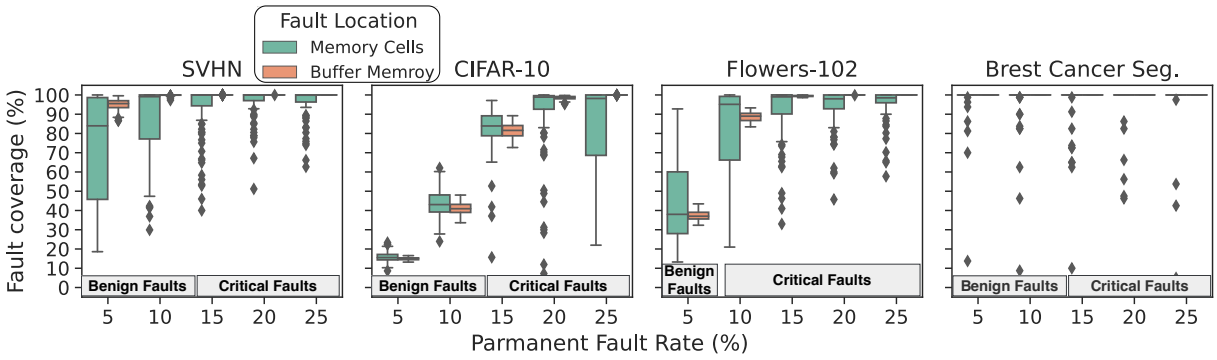


Figure 9.4.: Distribution of fault coverage when dealing with permanent faults in CiM architectures that affect memory cells storing weights and buffer memory storing activations across different datasets. **It is recommended to view this figure in color.**

sensitivity to permanent faults compared to other classification models. Specifically, the accuracy drops noticeably beyond the 5% fault rate. Therefore, faults at these fault rates are considered critical.

On the other hand, in the case of the segmentation task, there is a noticeable decrease in the area under the ROC curce (AUC) score around the 5% fault rate for both permanent and soft faults. Therefore, from 5% faults onward, faults are considered critical.

Similar accuracy drop patterns are observed for permanent and soft faults in the buffer memory of the CiM architectures that store activations. Also, we consider the CiM architecture as nonfunctional when the accuracy drops more than 20% below its baseline fault-free accuracy.

In the case of permanent faults of critical nature in memory cells and buffer memory that stores weights and activations, respectively, fault coverage consistently approaches 100% for all datasets, as shown in Fig. 9.4. In this scenario, the worst-case median of the fault coverage distribution is ~ 85% at the 15% fault rate for the CIFAR-10 dataset.

Similarly, in the case of multiple soft faults in the memory cells and buffer memory of the CiM architecture, it can be observed in Fig. 9.5 that the proposed method can consistently achieve 100% fault coverage in the presence of critical faults.

Although detecting critical faults is important, in some scenarios, detecting benign faults before they catastrophically impact the performance of the CiM architecture is also important. This is because many faults, such as retention faults, accumulate over time and can catastrophically impact accuracy in the future. As can be seen in Figs. 9.4 and 9.5, even for benign faults, our proposed method can achieve up to a 100% median of the fault coverage distribution.

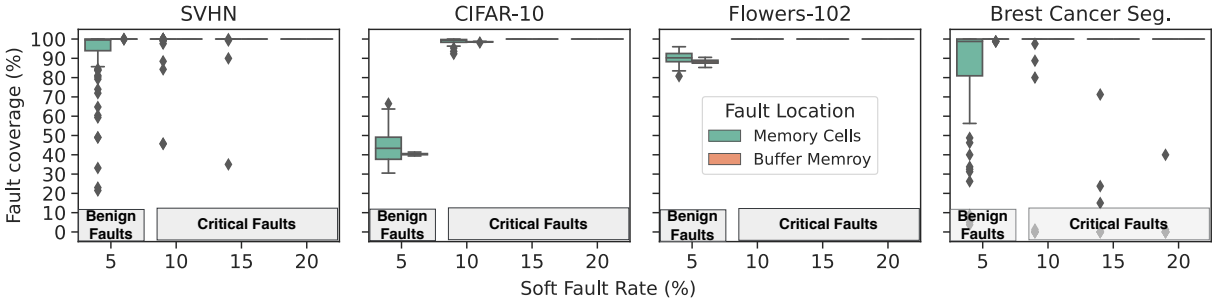


Figure 9.5.: Box plots depicting the distribution of fault coverage of the proposed method under soft faults on memory cells and buffer memory of CiM architectures. **It is recommended to view this figure in color.**

The change in the uncertainty fingerprint distribution depends on the fault rate and its impact on the performance of the CiM architecture. As shown in Fig. 9.7 (left), as the soft fault rate increases, the uncertainty fingerprint distribution moves further away from its baseline. The amount and direction of drift depend on the dataset, the NN topology, the location of the fault, and the type of fault. Therefore, even at the same fault rate, the fault coverage varies across NN topologies, datasets, and fault types. Also, since the accuracy loss at low fault rates is marginal at low fault rates (benign faults), coverage is also low. For instance, with a 5% permanent fault, the mean accuracy loss in CIFAR-10 is merely 2%, resulting in a coverage of around 20% at this fault rate. Given the minimal accuracy drop, such faults are deemed benign, rendering the low coverage relatively harmless.

In summary, on both permanent and soft faults affecting memory cells and buffer memories of CiM architectures, the proposed method can achieve a fault coverage distribution with a median of more than $\sim 85\%$ **for critical faults. In the event that the CiM architecture is not functional, a consistent 100% coverage can be obtained**. Furthermore, relatively high fault coverage can be achieved in the case of marginal degradation in accuracy due to benign faults. Note that the outliers in the box plots of Figs. 9.4 and 9.5 are also rare instances, and the fault coverage represents true positive cases. As the true positive rate is consistently high at 100%, our method can detect most critical faults accurately with rare false negative instances.

9.3.4. Analysis of Faults in the Uncertainty Head

Since the faults in the uncertainty head directly affect the uncertainty fingerprint, our approach can detect faults in the uncertainty effectively. Specifically, our approach can consistently achieve fault coverage of 98% to 100%. As shown in Fig. 9.6. Here, noise is injected into the memory cells that store weights of uncertainty head with increasing perpetuation rates.

9.3.5. Analysis of FPR and Comparison With Related Works

Earlier, we showed that the proposed method attains high fault coverage for both permanent and soft faults. However, maintaining low fault coverage in fault-free scenarios is also important to minimize false positive alarms. Specifically, our method exhibits a false positive rate of 11.17%, 9.72%, 7.25%, and 12.5% for the SVHN, CIFAR-10, Flowers-102, and Breast cancer segmentation datasets, respectively.

There is a trade-off between the true positive rate and the false positive rate. Calculating the boundaries l and h in the 1.5% and 97% quantiles of the uncertainty fingerprint distribution reduces the false positive rate to 8.65%, 6.8%, 5.75%, and 6.25%, respectively, but the true positive rate is reduced by $\sim 2\%$. Regardless, our proposed method requires a 62.2% and 89.35% lower false positive rate to achieve a 100% true positive rate compared to open-set recognition-based fault detection [119]. Specifically, their work requires a false positive rate of $45 - 69\%$ and $97 - 98\%$ for CIFAR-10 and SVHN, which is unacceptable for many

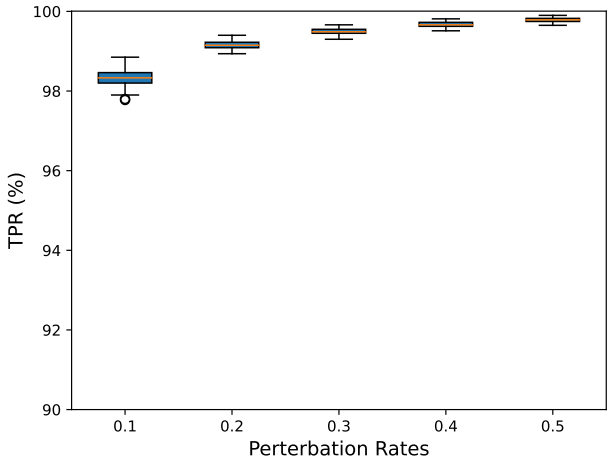


Figure 9.6.: Distribution of fault coverage due to faults in the uncertainty head. Random noise with increasing fault intensity is injected into the memory cells that store the weights of the uncertainty head.

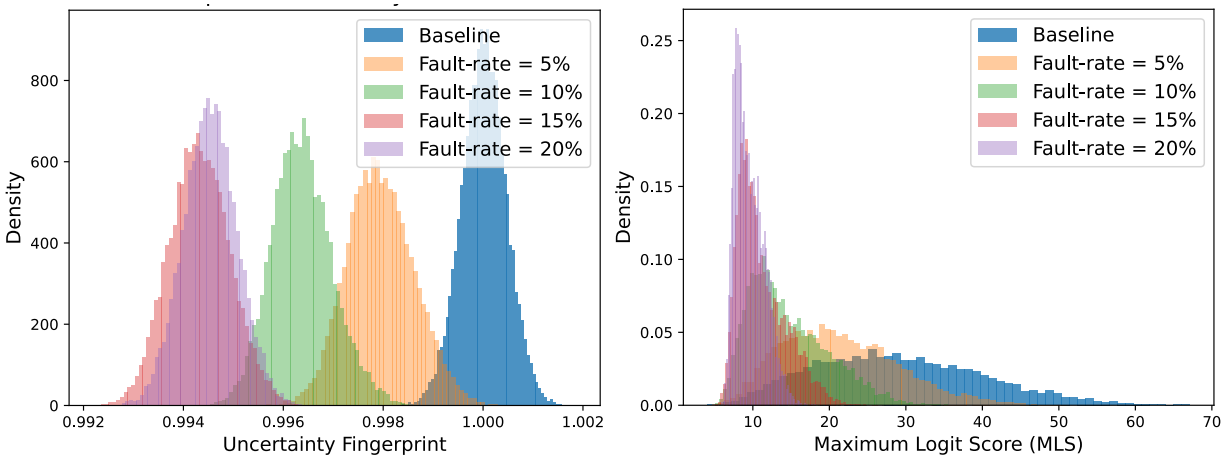


Figure 9.7.: Change in the distribution of the proposed uncertainty fingerprint and maximum logit score [234] method due to soft faults on the memory cells of CiM architectures. **It is recommended to view this figure in color.**

applications, as it would raise too many false alarms about the state of the model. Also, as shown in Fig. 9.7, the distribution of the maximum logit score [234], an open set recognition method studied in [119], overlaps at most fault rates, complicating fault detection.

9.3.6. Discussion

9.3.6.1. Detecting Faults in the Prediction Head

Although our method has been shown to be effective in detecting different types of faults in different locations of the CiM architecture, it cannot detect faults in the prediction head. Faults in the prediction head are equally likely as in other layers of the NN. However, their impact can be more significant, as their output directly becomes the model prediction.

Therefore, to detect faults in the prediction head, we propose optional feedback from the prediction head to the uncertainty head. This allows us to test the faults in the main head. The uncertainty fingerprint can be defined in this case as:

$$\text{Uncertainty Fingerprint} = \frac{1}{2} \times (\mathcal{U} + \max(\hat{y}) \tag{9.2}$$

Here, $\hat{\mathbf{y}}$ represents the softmax score from the prediction head, and thus $\max(\hat{\mathbf{y}})$ represents the predicted class probability. In addition, $\mathcal{U}$ represents the maximum value of the uncertainty head, as described earlier.

With the updated uncertainty fingerprint, we have found that our approach is able to detect faults in the prediction head with up to 100% coverage. Also, fault coverage for faults in the backbone of the model can also be improved by as much as 2% to 5%. For example, the mean coverage of ResNet-18 topology on a 5% bit-flip faults (benign) increases to 46.87% from 44.15%. However, the false positive rates can increase. For example, for ResNet-18 topology, the false positive rate increases to 12.95% from 9.25%.

9.3.6.2. Improving Fault Coverage Further

While occasional false positives are not severely harmful, a lack of coverage is. Therefore, the bounds l and h can be adjusted to favor the detection coverage, even if it slightly increases the FPR, balancing between detection coverage and false positive rate.

9.3.7. Scientific Impact of This Work

The scientific impact of this work is summarized as follows:

1. **Uncertainty Fingerprint Approach**: We introduce the *uncertainty fingerprint* approach, which allows for runtime status checking of AI accelerators by matching offline fault-free fingerprints during online operation. Such fingerprinting is an attractive research direction for fault detection.
2. **Dual-Head Neural Network Topology**: The proposed dual-head NN topology allows simultaneous monitoring of the fault status of the edge AI accelerators, eliminating the need for extra forward passes. Our proposed topology can be extended for other applications where concurrent monitoring of NN status is required.
3. **Scalability**: Although our approach is demonstrated using CiM architectures, the methodologies are applicable to any AI accelerator architecture and NN topologies.
4. **Applicability in Real-Time AI**: Our proposed approach allows concurrent self-testing without interruption, making it suitable for real-time applications where system interruption is impractical.

9.3.8. Section Conclusion

In this section, we propose a novel approach to the concurrent self-test of NNs using the dual-head model and uncertainty fingerprint. Our approach enables continuous fault monitoring without necessitating the cessation of the primary task, thus fulfilling a critical gap in most of the current research. The proposed dual-head NN topology is specifically designed to produce uncertainty fingerprints and primary predictions in a single shot. During online operation, our approach can concurrently self-test CiM architectures with up to 100% coverage while maintaining the performance of the primary task in the benchmark datasets and topologies.

10. Disentanglement of Source of Uncertainty

Conventional fault detection and uncertainty estimation approaches cannot disentangle the sources of uncertainty, e.g., the uncertainty of the prediction is due to data distribution shifts or faults in the edge AI accelerator. Knowledge of sources of uncertainty can aid in targeted uncertainty mitigation strategies, such as hardware maintenance in terms of fault removal, data augmentation, or re-training. This section proposes a solution for disentangling the source of uncertainty. Our approach complements the method from the previous chapter and is based on our paper [233].

10.1. Problem Statement

In dynamic real-world deployment of NNs, various sources of uncertainties at the input, e.g., sensor noise, and hardware faults, e.g., soft or permanent faults, are feasible. In runtime (concurrent) testing of AI accelerators that monitor or analyze the NN output or intermediate results, it is usually assumed that the data are from the same distribution as training data. Therefore, it is assumed that the change in the NN output or intermediate results is caused by faults and variations. However, the change can be caused by out-of-distribution data. In this case, a concurrent testing method cannot disentangle the source of uncertainty. For example, in our previous work that was presented in Chapter 9, if the runtime fingerprint of the proposed dual head model does not match, it could be because of out-of-distribution data or faults in the CiM architectures. To further improve the reliability and robustness of NNs, it is crucial to *disentangle the sources of uncertainty* so that targeted uncertainty mitigation techniques can be applied. Knowledge about the sources of uncertainty aids in targeted uncertainty mitigation strategies, whether that involves hardware maintenance, data augmentation, or re-training followed by re-deployment.

Therefore, we propose a methodology to disentangle the sources of uncertainty. Our approach complements our previous work that was presented in Chapter 9 to improve the overall reliability.

10.1.1. Methodology

In our approach, once the uncertainty fingerprint does not match, we perform explicit testing with a predefined single disentanglement test vector. The disentanglement test vector is carefully selected from the training data with a known fault-free fingerprint that matches the ideal fingerprint (maximum value of one) described in our work in Chapter 9.

For explicit testing, the disentanglement test vector is forward passed through the model, and the fingerprint of the model is compared to the known fingerprint. If they match, it suggests that the original uncertainty is due to out-of-distribution data or soft faults lasting only one cycle, e.g., faults that are no longer present in the system. Following that, an expert, e.g., a human or a more sophisticated model, such as a Bayesian NN, further classifies the original input as in-distribution or out-of-distribution and provides the correct label for the input. Identification of in-distribution data can indicate the presence of soft faults, assuming the model did not make a false alarm. Nonetheless, if the expert confirms OOD classification, this validates the uncertainty detection of the proposed method while still suggesting potential soft faults.

However, if the fingerprint of the model given the test vector does not match the known fingerprint, it implies the presence of permanent faults or soft faults lasting several cycles. Once again, an expert

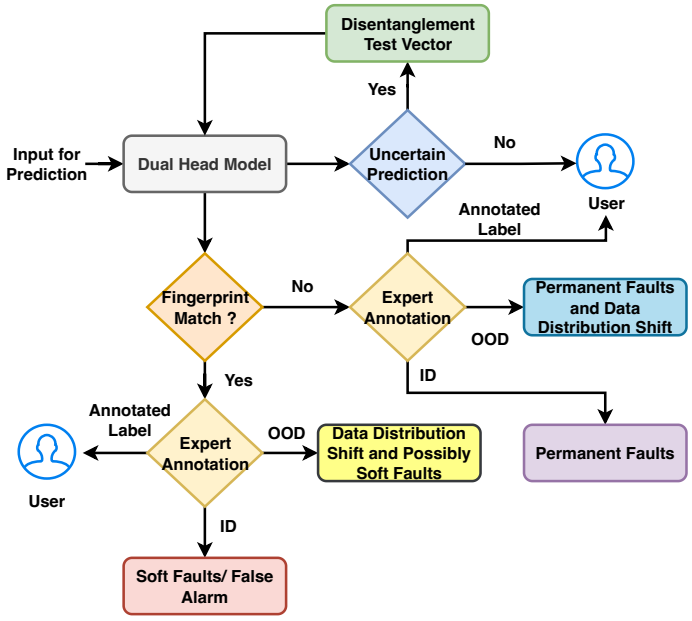


Figure 10.1.: Flow diagram for the disentanglement of the sources of uncertainty. In the case of permanent faults, an uncertainty reduction approach should be applied.

annotator classifies the input as in-distribution or out-of-distribution. An in-distribution classification confirms permanent faults in the CiM architectures, while an out-of-distribution classification indicates both permanent faults and out-of-distribution input at the same time.

Our method effectively disentangles most of the uncertainty sources. However, it is limited to permanent faults and soft faults lasting several cycles. It is difficult to isolate soft faults of extremely short duration, as they do not persist in the CiM architecture long enough to be detected by our test vector approach. The overall flow diagram of our approach is depicted in Fig. 10.1.

10.2. Results

10.2.1. Evaluation Setup

To evaluate the capability of the proposed approach in disentangling the source of uncertainty, we performed five scenarios. Specifically,

1. Input Noise (Case 1): Gaussian noise is injected into the input with 50% probability. That is, each input is either Gaussian noise or a clean input with a probability of 50%.
2. Short-Duration Soft Faults (Case 2): Faults are injected into the buffer memories that store intermediate activation of the NNs while performing the prediction.
3. Permanent Faults (Case 3): Faults are injected into the memory cells that store the weights of the NNs.
4. Input noise + permanent faults (Case 4): Gaussian noise is injected into the input with a probability of 50%, and stuck-at faults are injected into the memory cells that store the weights of the NN.
5. Input noise + Soft Faults (Case 5): Gaussian noise is injected into the input with a probability of 50%, and faults are injected into the buffer memories that store the intermediate activation of the NN.

All out-of-distribution data are labeled for evaluation, and we present the percentage of disentanglement with our proposed approach.

Table 10.1.: Evaluation of the disentanglement of the sources of uncertainties. Five scenarios with isolated and a combination of sources of uncertainties are explored.

Topology	Dataset	Average	Disentanglement Scenarios						
			Case 1	Case 2	Case 3	Case 4		Case 5	
			Input	Permanent Faults	Soft Faults	Input	Permanent Faults	Input	Soft Faults
VGG	Flower-102	100%	100%	100%	100%	100%	100%	100%	100%
ResNet-18	CIFAR-10	99.85%	100%	100%	100%	99.33%	100%	99.67%	100%
PreActResNet-34	SVHN	99.72%	100%	100%	100%	98.07%	100%	100%	100%
U-Net	Brest Cancer Seg.	98.57%	100%	100%	100%	90%	100%	100%	100%

10.2.2. Analysis of Disentanglement of Source of Uncertainty

As shown in Table 10.1, our proposed method shows a high degree of accuracy in disentangling sources of uncertainty in all the topologies and uncertainty scenarios evaluated. In most cases, our approach achieves 100% accuracy in disentangling uncertainty sources. Specifically, for Cases 1, 2, and 3 (isolated sources of uncertainty), our approach demonstrates perfect disentanglement for input noise, permanent faults, and short-duration soft faults. However, in case 4 where input noise and permanent faults are combined, it leads to a minor reduction in accuracy, e.g., 98.07% on PreActResnet. Similarly, for case 5, minor reductions in accuracy are observed. The reductions are attributed to the compounding effects of multiple sources of uncertainty and the fact that some of the inputs escape the model even though they are out of distribution.

Therefore, the proposed method consistently demonstrates its ability to accurately disentangle various sources of uncertainty. This significantly enhances the reliability of the CiM architecture and highlights its potential for deployment in complex and dynamic environments.

10.3. Scientific Impact of This Work and Contributions

The contributions of this paper are summarized as follows:

1. **Disentanglement of Uncertainty Sources**: Our work underscores the need for disentangling the source of uncertainty. This capability is crucial for root cause analysis, e.g., distinguishing whether prediction uncertainty arises from data distribution shifts or hardware faults, enabling targeted mitigation strategies.
2. **Integration with Concurrent Testing**: The proposed methodology can be integrated seamlessly with any concurrent testing frameworks and predictive uncertainty estimation method. It ensures that AI accelerators can be continuously monitored and faults can be detected without interrupting their primary tasks.
3. **Scalability and Adaptability**: The proposed methodology is designed to be scalable across various edge AI accelerator architectures, neural network topologies, datasets, and tasks. This scalability makes the approach broadly applicable and adaptable to diverse AI applications and hardware configurations.

10.4. Chapter Summary

This chapter proposes a methodology to disentangle the sources of uncertainty. Our approach complements the work presented in the previous section to improve the overall reliability. Our method effectively identifies whether the predictive uncertainty is due to out-of-distribution data or hardware faults by

using predefined disentanglement test vectors and comparing the resulting fingerprints with the fault-free fingerprints. The evaluation of the proposed method shows high accuracy in distinguishing between different sources of uncertainty. Therefore, it significantly enhances the reliability of AI systems in complex and dynamic environments.

Part III.

Methods for Uncertainty Reduction

To reiterate, AI accelerators are prone to non-idealities such as manufacturing and in-field defects and variations, as well as environmental factors such as temperature fluctuations. We have presented several scalable and efficient methods for estimating uncertainties in edge AI accelerators using online functional testing. However, reducing uncertainties due to hardware non-idealities via fault tolerance for functional safety is crucial, especially in safety-critical applications. Specifically, since the inference accuracy of NNs on edge AI accelerators is negatively impacted, we aim to increase the accuracy despite faults or variations in weights or activation. However, our aim is to propose methods that have low hardware overhead so as not to increase the burden on already resource-constrained edge AI accelerators.

This thesis targets uncertainty reduction with self-healing and runtime periodic maintenance approaches. Self-healing approaches, also called implicit fault tolerance, are designed to be inherently fault-tolerant without needing any run-time maintenance. Specifically, it performs training and design time optimizing for graceful degradation in accuracy due to faults and variations. That means the rate of accuracy degradation is significantly lower in comparison to baseline. Also, self-healing approaches by design do not need to pause the operation of the system for uncertainty reduction. On the other hand, runtime maintenance approaches require run-time intervention, which can be periodic or can be triggered by an online testing approach.

This part presents all our solutions to the above-mentioned challenges.

11. Self-Healing Approaches

Self-healing allows the edge AI accelerator to continue operating without interruption to reduce uncertainty. It ensures that edge AI accelerators can maintain high performance and reliability in dynamic environments without the need for frequent maintenance or system pauses, making them ideal for critical applications where continuous operation is essential.

This Chapter presents a self-healing approach for manufacturing and run-time variations for the entire operating temperature range (up to 120°C). Then, we present a self-healing approach for a Bayesian neural network where we designed a normalization and Dropout layer for implicit fault tolerance.

11.1. Self-Healing the Impact of Manufacturing and Infield Thermal Variations

This section proposes a self-healing approach for memristor-based CiM architectures. To reiterate, CiM architectures mitigate the data flow between processing and memory units. However, the manufacturing process and temperature fluctuations at runtime have a huge impact on the calculation of the activations of a layer and reduce the overall post-mapping inference accuracy of the NN.

To deal with the process and runtime variations, existing solutions either only mitigate process variations and cannot deal with run-time variations [125, 126, 127], or require costly (on-the-fly) re-mapping and re-training based on sensed temperature [24, 121, 122].

In this section, we propose a self-healing solution to mitigate the impact of process and temperature (runtime) variations on the inference accuracy of NNs mapped to STT-MRAM-based CiM architectures. However, our approach is applicable to other memristor technologies, such as Redox-based Random Access Memory (ReRAM) and Phase Change Memory (PCM).

We targeted STT-MRAM because, among memristor technologies, STT-MRAM has reached a comparable mature state as indicated by many industrial adaptions [235, 236, 237]. Furthermore, the switching speed and the endurance of STT-MRAM are higher than those of other eNVM technologies [238]. Although STT-MRAM offers many benefits, their ON/OFF ratio is significantly lower than that of other memristor technology and has a comparably smaller sense margin. This is further exacerbated by temperature fluctuations at runtime. Even small variations in the conductance of the cells can lead to incorrect neuron activations.

This section aims to address this and is based on our conference and journal papers IEEE ETS23 [239] and ACM JETC [67].

11.1.1. Problem Definition

Due to the non-idealities of STT-MRAM, such as process and temperature variations, the partial sum current I_{ps} in the bit-line of the crossbar is also subject to variation and can be represented by a distribution rather than a fixed value as shown in Fig. 11.1(a). The distribution of the partial sum current I_{ps} and the sense margin of each partial sum state depends on the number of word-lines activated concurrently (m_{wl}) and the process variation (static variation). The sensing margin of I_{ps} is reduced, and the size of the overlapping region between states increases when more word-lines are concurrently activated (as shown in Fig. 11.1(b)) or the process variation is high. Due to the small ON/OFF ratio in STT-MRAM, the

reduction of the sense margin is more severe in this technology compared to other NVM technologies. As a result, the miss-quantization rate of the sensing circuits will also be higher.

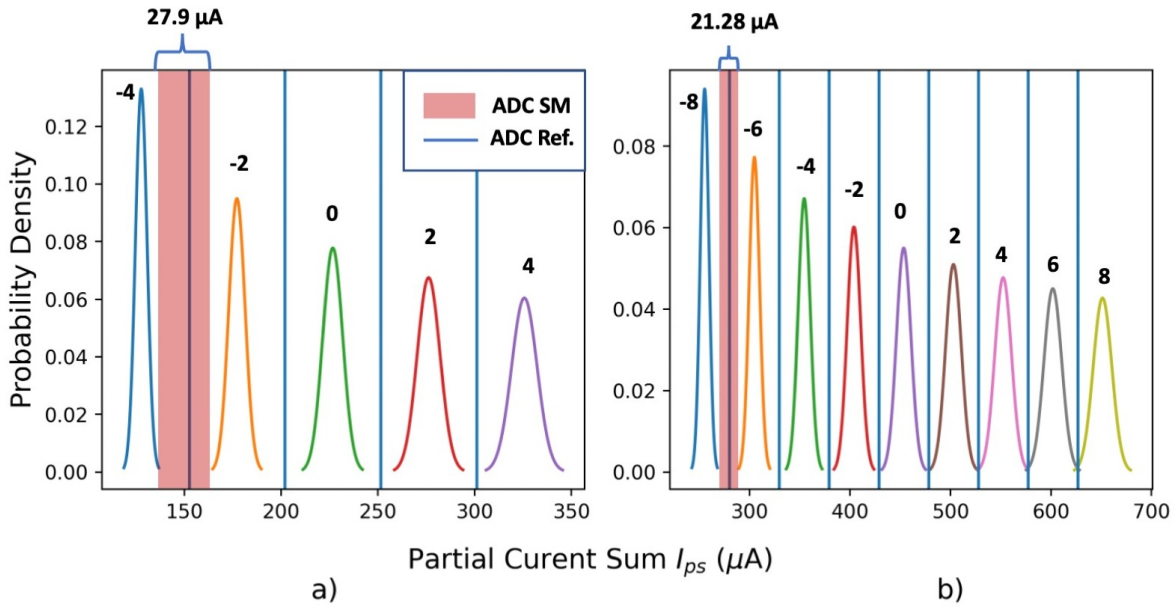


Figure 11.1.: (a) The distribution of possible partial sum currents (I_{ps}) depending on cell state combinations ($-4, -2, \cdots, 4$) when four word-lines are activated concurrently. Since weights and activations are binary, values such as -3 are not possible, (b) when more (m_{wl}=8) word-lines are activated concurrently, the sensing margin of ADC (ADC SM) becomes smaller. **It is recommended to view this figure in color.**

Furthermore, the distribution of I_{ps} can dynamically change with the temperature (dynamic variation), and hence, the size of the overlapping region increases further with temperature as shown in Fig. 11.2 (a). Because of the low ON/OFF ratio of the MTJs, even a small change in the conductance distribution can increase the bit-line current miss-quantization rate. As a result, the post-mapping inference accuracy can still degrade significantly even with a larger average sense margin, for instance, with a single I_{ref} as shown later in Section 11.1.3.

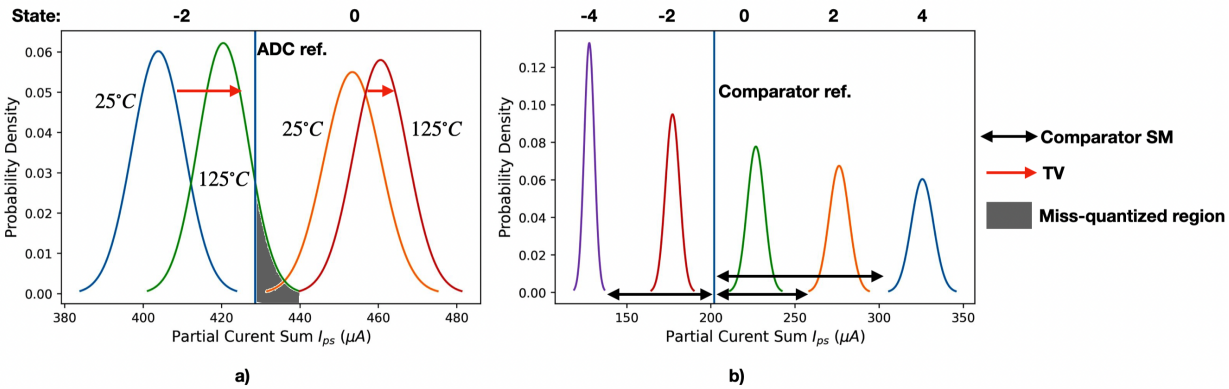


Figure 11.2.: a) Distribution shift of partial sum current (I_{ps}) for states -2 and 0 due to temperature variations (TV). The operating temperature increased from $25°C$ to $125°C$. b) quantized I_{ps} with a larger sense margin (SM) for some states. Partial sum currents more than I_{ref} are quantized to +1, while lower currents are quantized to -1. **It is recommended to view this figure in color.**

In practice, the process variations also affect the reference circuit. In other words, I_{ref} itself is subject to variations. The effect of I_{ref} variations should also be considered, as it can lead to degradation in inference accuracy.

Moreover, the training dataset can contain anywhere from a few thousand to millions of samples. For instance, the CIFAR-10 dataset has 50k samples. Finding T_b using all samples in the dataset will be too costly.

11.1.2. Methodology

For efficient inference of the NN with the STT-MRAM-based CiM architecture, referred to as STT-CiM in the following, each partial sum current I_{ps} can be quantized to +1 or −1 [65]. This allows sensing each partial sum with a comparator instead of a much more expensive ADC [240].

We have found that this quantization can implicitly increase the sense margin, as shown in Fig. 11.2(b). Therefore, this method can provide robustness against static process variation. However, it is not sufficient for dynamic temperature variation. Since the ON/OFF ratio of the MTJ decreases even further with a higher temperature, this will lead to the miss-quantization of partial current sum I_{ps}, as shown in Fig. 11.2(a). Figs. 11.1 and 11.2, are generated according to the simulation setup described later in Section 11.1.3.1 and the MTJ parameters reported in Table 11.1.

Therefore, we propose a temperature-aware design-time approach for generating the optimum I_{ref}, which can self-heal the effects of temperature-induced partial sum current distribution on the inference accuracy.

Generally, there are $m_{wl} + 1$ possible partial sum states,

$$-m_{wl}, -m_{wl} + 2, \cdots, 0, \cdots, m_{wl} - 2, m_{wl}$$

, separated by 2 when m_{wl} number of word-lines are activated concurrently. Quantizing the partial sum without any training modification will lead to a significant degradation in the inference accuracy, due to the loss of information that the next layer receives. The post-training network reconstruction approach proposed in [65] introduces additional neurons for the quantization, which results in a memory overhead and requires retraining to regain the accuracy loss due to the quantization.

Hence, we propose a training algorithm to consider the quantization of each partial sum during training with a straight-through estimator (STE) [241] that does not introduce any extra runtime overhead or requires re-training.

Our proposed method first quantizes each partial sum during offline training, as discussed in Section 11.1.2.1. Then we generate the optimal reference current $\hat{I_{ref}}$ (one time only) for the entire operating temperature range, as discussed in Section 11.1.2.2. The NN is deployed for the inference with the optimal $\hat{I_{ref}}$, which does not require re-mapping or re-training.

11.1.2.1. NN Training Modification for I_{ps} Quantization

Our modified training algorithm applies the quantization operation in two stages. First, the weight matrix is binarized as follows:

$$\text{sign}_W(\mathbf{W}) = \begin{cases} \frac{1}{m_{wl}}, & \text{if } x \geq 1 \\ \frac{-1}{m_{wl}}, & \text{otherwise.} \end{cases}$$

Therefore, each partial sum a^s for a step s, before binarization, has a value between -1 and $+1$, $a \subset [-1, 1]$. Then, to implement the quantization of each partial sum, the sign$(\cdot)$ function is applied to each partial sum during the inference (in the forward pass). The calculation of a partial activation can be expressed as

$$\mathbf{z}_l^s \leftarrow \text{sign}\left(\mathbf{z}_{l-1}^{m_{wl},s} \oplus \mathbf{W}_l^{m_{wl},s}\right),$$

where $\mathbf{W}_l^{m_{wl},s}$ is the weight sub-matrix, $\mathbf{z}_{l-1}^{m_{wl},s}$ is the input sub-vector, and $\mathbf{z}_l^s$ is the resulting partial activation for a single step s with the activated word-lines m_{wl} and layer l. The modified $\text{sign}_w(\cdot)$ function for binarizing the weight matrix combined with binarizing each partial activation adds a small noise to each neuron, for instance, $0.9 \rightarrow +1$.

The gradients required for the back-propagation algorithm [242] cannot flow through the sign(x) function, since the derivation of the sign$(\cdot)$) function is zero. Therefore, the traditional gradient descent-based learning algorithm [243] cannot be applied during backward propagation to update NN parameters. To still obtain gradients for learning, an STE is used [241]. In STE, the gradient of the quantization $\text{sign}_w(\cdot)$ function is treated as an identity in the backward pass and can be expressed as:

$$\delta STE(\mathbf{z}) = \frac{\delta\, \text{sign}_w(\mathbf{z})}{\delta \mathbf{z}} = \begin{cases} +1, & \text{if } \mathbf{z} \geq -1 \quad and \quad \mathbf{z} \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$

Fig. 11.3 graphically illustrates the process of learning quantized partial sums. Hence, (nonzero) gradients

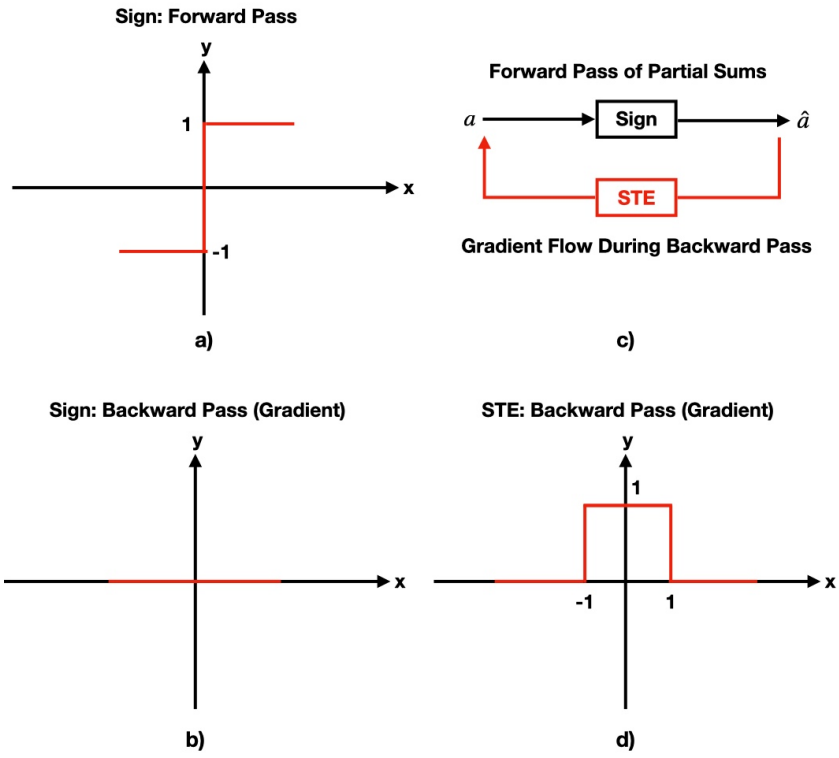


Figure 11.3.: a) Forward pass of sign_w function, b) illustrate the zero-derivative problem of sign_w, c) learning of quantized partial sum with STE, and d) gradient of STE during backward pass.

can be obtained during the back-propagation. Furthermore, real-valued activations and weights are used during back-propagation and gradient calculation.

In a single step s, a group of m_{wl} word-lines are activated concurrently, and a partial sum is calculated for this step. It takes total $\bar{S} = \frac{m}{m_{wl}}$ steps to calculate the final activation of a layer, which becomes the input for the next layer. The overall training algorithm is described in Algorithm 5.

Algorithm 5 Proposed training algorithm for binarizing each partial sum

Require input $\mathbf{x}$, number of steps $\bar{S}$, number of layers L, and BatchNorm$(\cdot)$ parameters β, and γ

$\mathbf{z}_1 \leftarrow \mathbf{x}$
1. first and hidden layers
for $l = 1$ to $L-1$ **do**
  $\mathbf{z}_l^s \leftarrow 0$
  for $s = 1$ to $\bar{S}$ **do**
    $\hat{\mathbf{W}}_l \leftarrow \text{sign}_W(\mathbf{W}_l)$ ▷ Binarize weight matrix
    $\mathbf{z}_l^s \leftarrow \mathbf{z}_{l-1}^s + \text{sign}\left(\mathbf{z}_l^{m_{wl},s} \oplus \hat{\mathbf{W}}_l^{m_{wl},s}\right)$ ▷ MAC operation for a single step s
  end for
  $\mathbf{z}_l \leftarrow \text{sign}(\text{BatchNorm}(\mathbf{z}_l^s, \beta_l, \gamma_l))$
end for
2. Output Layer
$a_L^s \leftarrow 0$
for $s = 1$ to S **do**
  $\hat{W}_L \leftarrow \text{sign}_W(W_L)$ ▷ Binarize weight matrix
  $a_L^s \leftarrow a_L^s + \text{sign}\left(a_{L-1}^{m_{wl},s} \oplus \hat{W}_L^{m_{wl},s}\right)$ ▷ MAC operation for a single step s
end for
$a_L \leftarrow \text{BatchNorm}(a_L^s, \beta_L, \gamma_L)$

11.1.2.2. Design-time Reference Current Generation

In STT-CiM operation, the reference current I_{ref} for the sensing circuitry needs to be calculated based on the conductance distribution of the LRS and HRS of the MTJ device. However, if I_{ref} is generated at one temperature, for example, at room temperature ($25°C$), it is not optimized for the possible operating temperature range. Consequently, the size of the overlapping region of the I_{ps} distribution can increase.

To allow self-healing of CiM architectures , we propose a one-time generation of a temperature-aware reference current at the design time $\hat{I_{ref}}$, which can deal with the variations due to temperature throughout the operating temperature range of the device. The proposed method first performs a design time analysis to find the temperature to which the inference accuracy does not fall below a predefined threshold with a binary search algorithm with lower worst-case search cost $O(\log_2 \bar{H})$. We refer to this temperature as the boundary temperature (T_b), and $\bar{H}$ is the number of steps or upper bound of the search space. Since T_b is likely to be in the mid-region of the overall operating temperature range rather than at extreme ends, the overall search cost will be lowered due to the lower initial search space. Note that as the temperature increases from low to high, sorting the search array, which is generally necessary for the binary search algorithm, is not necessary.

The boundary temperature is estimated based on training data. Therefore, as mentioned earlier, this can lead to significant overhead if all samples in the dataset are used. To reduce this overhead, we propose a small subset of the training or validation dataset, e.g., 20%, to be used to find T_b. Once T_b is determined, the reference current $\hat{I_{ref}}$ is generated at the boundary temperature T_b.

In addition, it can be very costly to perform the proposed method throughout the operating temperature range. Therefore, we propose increasing the operating temperature in steps (R) of $5°C$, $10°C$, or $20°C$ since our goal is to find T_b approximately rather than exactly. However, it is not required to consider the integer temperature or the larger temperature steps to find T_b. A regression plot, as shown in Fig. 11.4, can be used to determine the conductance (reciprocal of resistance) distribution of the MTJ device with P and AP states at non-integer temperatures.

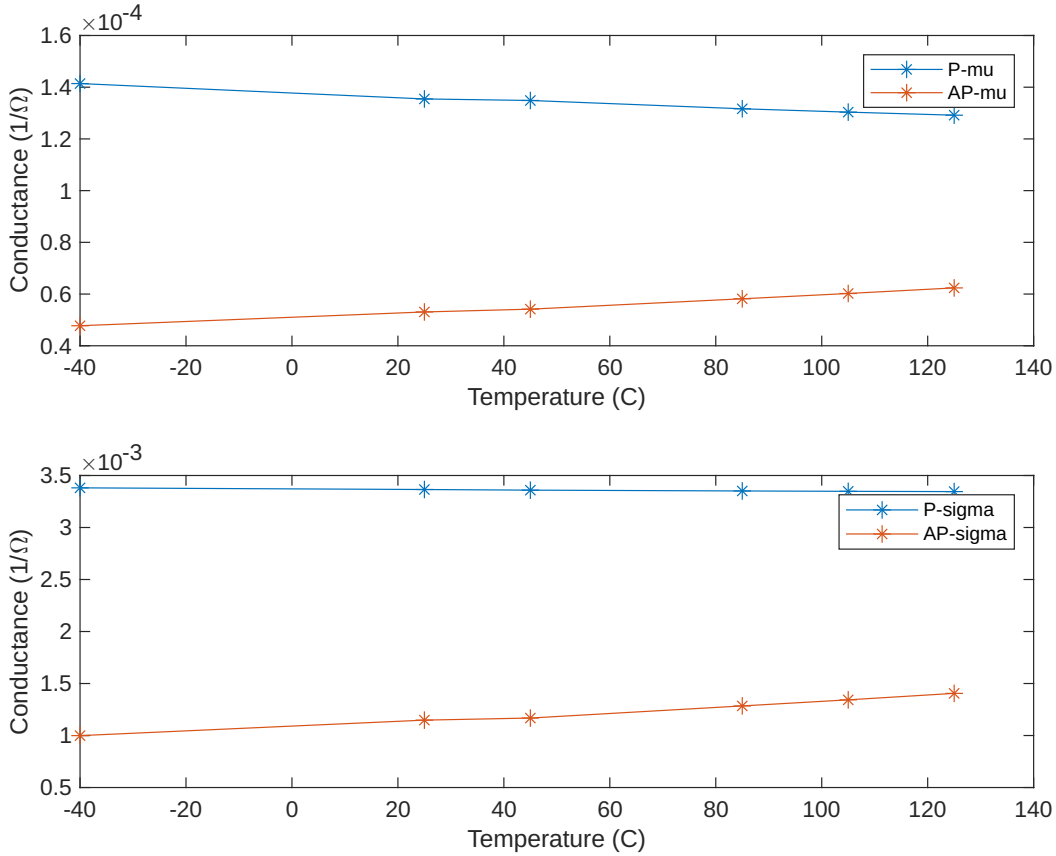


Figure 11.4.: Regression plot showing the distribution of P and AP states of MTJ at various temperatures. **It is recommended to view this figure in color.**

The intuition behind our approach is that generating $\hat{I_{ref}}$ at T_b will lead to fewer miss-quantizations of I_{ps} and, in turn, self-healing the impact of variations. Consequently, a graceful degradation of the inference accuracy. Furthermore, the offset of $\hat{I_{ref}}$ at high temperature ($125°C$) is significantly lower. The overall algorithm for the generation of $\hat{I_{ref}}$ is shown in Algorithm 6. The function $\mathcal{F}(.)$ represents the forward propagation of the NN. Our proposed method assumes that all STT cells are at the same high temperature. Please note that our proposed method generates I_{ref} using modeled device distributions at different temperatures during the design time, and therefore, no post-mapping hardware retraining is performed.

In Algorithm 6, the reference current for the proposed temperature search must be initialized at the beginning. The temperature at which the initial I_{ref} needs to be generated in the Algorithm 6 is determined according to the equation $T \leftarrow M \times R + T_{init}$. However, note that the initial I_{ref} can be generated at other temperatures according to the lower end of the expected operating temperatures. A different value can be chosen for the lower T_{init}, upper T_{max}, and temperature search step R rather than the one used in the Algorithm 6.

Note that the proposed Algorithm 6 for finding the boundary temperature does not require gradient information for temperature search, e.g., backpropagate based on a cost function, since it is evaluated on a pre-trained model and in the inference state. Furthermore, the determined T_b will vary depending on parameters of the Algorithm 6, e.g., the value chosen for R. To this end, it can be said that there are multiple minima of the Algorithm 6. However, this phenomenon is common for any optimization algorithm. For example, the minima for the loss function in NN optimization depends on various aspects such as the learning rate and the number of iterations.

Algorithm 6 Proposed algorithm for temperature-aware I_{ref} generation at design time

Require pre-trained NN from Algorithm 5, m_{wl}, Dataset $\mathcal{D}$, Threshold th, predicted label y, correct label y', mean μ_0 and μ_{-2} of $\mathbf{z}^s = -2$ and 0 state's distribution if WL is even else $\mathbf{z}^s = -1$ and 1 state's distribution

$\mathcal{D}_k \subset d$ ▷ Select a subset of the dataset to find the boundary temperature
$T_b, R, T_{init}, T_{max} \leftarrow 0, 20, 25, 125$ ▷ Set boundary temperature and other variables
$\bar{L}, \bar{H} \leftarrow 0, \frac{T_{max}-T_{init}}{R}$ ▷ Set lower-bound ($\bar{L}$) and upper-bound ($\bar{H}$)
while $\bar{H} >= \bar{L}$ **do**
  $M \leftarrow L + \text{ceil}(\frac{\bar{H}-L}{2})$ ▷ Compute midpoint (Mid)
  $T \leftarrow \text{Mid} \times R + T_{init}$ ▷ Compute temperature for reference generation
  $y \leftarrow \mathcal{F}(\mathcal{D}_k)$ ▷ Compute predicted label
  $A \leftarrow (\sum y == y')/size(\mathcal{F})$ ▷ Compute inference accuracy, A
  if $A >= th$ **then**
    $T_b \leftarrow T$
    $\bar{L} \leftarrow \text{Mid}$ ▷ Update lower bound
  else
    $\hat{I_{ref}} \leftarrow 0.5 \times (\mu_{-2} + \mu_0)$ ▷ Generate $\hat{I_{ref}}$ at T_b
  end if
end while

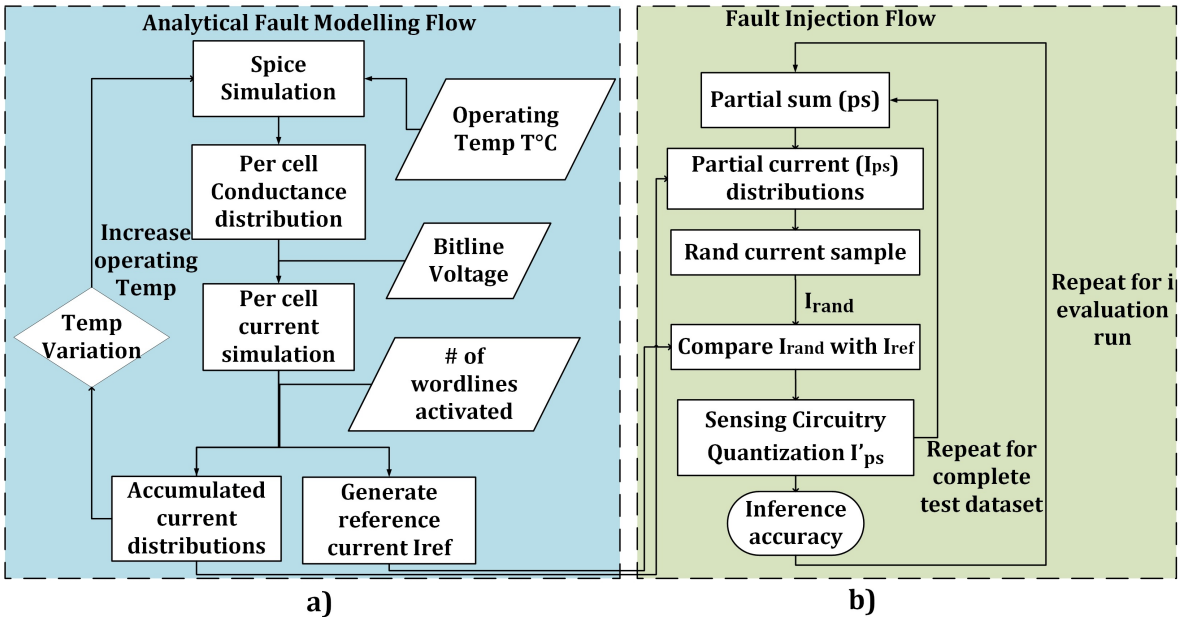


Figure 11.5.: a) Overview of analytical fault modelling, and b) fault injection flow.

11.1.3. Results

11.1.3.1. Evaluation Setup and Imperfection Injection Framework

We have used a PyTorch [244] based custom build simulation framework to analyze the impact of non-ideal circuit and device properties on the inference accuracy of the STT-CiM operation. The overall analytical fault model and the fault injection flow are shown in Figs. 11.5 (a) and (b), respectively.

Initially, for static process variation, the conductance mean (μ_{STT}) and the standard deviation (σ_{STT}) of the MTJ devices are obtained by performing the electrical level simulation with SPICE under room temperature conditions ($25°C$). Then, to get the possible probability density function (PDF) of each possible I_{ps} state, the r PDFs of LRS and the $m_{wl} - r$ PDFs of HRS are summed up, where $r = 0, 1, \cdots, m_{wl}$. However, for

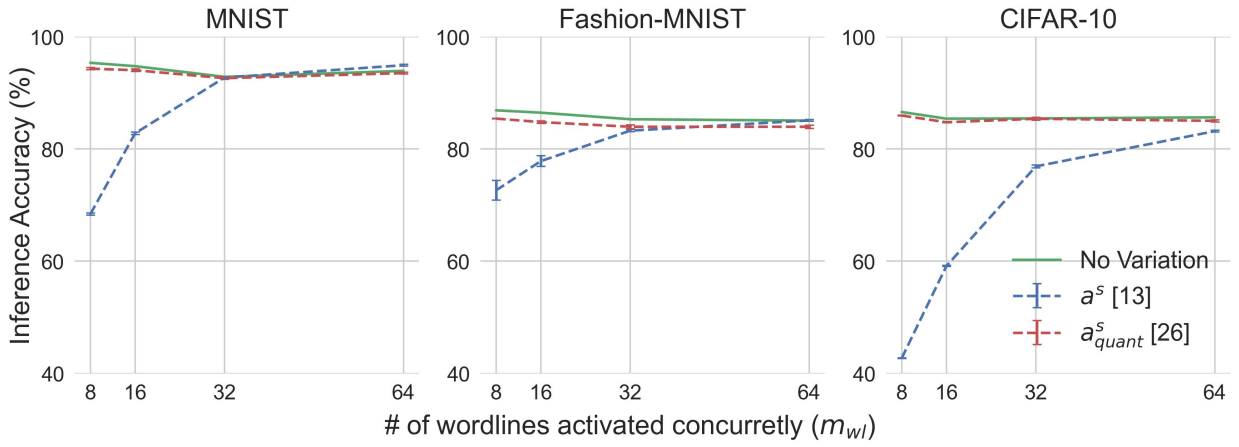


Figure 11.6.: Impact of process variation on the inference accuracy of the MNIST, Fashion-MNIST and CIFAR-10 datasets considering process variation only for different concurrently activated word-lines. Here, linear layers of the quantized a^s_{quant} [65] and unquantized a^s [126] partial sum NN model are considered. **It is recommended to view this figure in color.**

dynamic temperature variation, the mean (μ) and the standard deviation (σ) of the conductance of the MTJ device are obtained for each operating temperature by SPICE simulation. Then, the PDF of each possible I_{ps} is calculated. The electrical simulation set-up shown in Table 11.1 is based on work in [245].

Table 11.1.: MTJ parameters and simulation setup. The MTJ parameters are based on work [245]

Parameter	Value	Parameter	Value
VDD	0.8V	MTJ radius	20nm
Nominal Temperature	25°C to 125°C	RA	$7.5\Omega\mu m^2$
CMOS Library	Globalfoundries 22FDX	TMR @ 0V	220%
Free/Oxide layer thickness	1.3/1.48 nm	'AP'/'P' resistance	19 kΩ / 6 kΩ
Process var. (AP/P)	4.7% / 4.05%		

The references I^{ADC}_{ref} of the flash-type ADC [126] are generated by dividing the possible current range ($I_{end} - I_{start}$) by the number of possible partial sum states, and the $\hat{I_{ref}}$ of the 1-bit sense amplifier [65] is calculated using Algorithm 6 with 20% of the training data. To simulate dynamic temperature variations, the reference current of the baseline ADC and the 1-bit sense amplifier [65] are generated at 25°C, as in a conventional design. Subsequently, the I_{ref} was not optimized for the entire operating temperatures as opposed to the proposed approach. For our approach, the reference current is generated according to Algorithm 6. Therefore, to distinguish between the baseline method for the generation of I_{ref} and the proposed method, we have simulated them with two reference currents. For our proposed methods, the reference current is optimized for the entire operating temperature, but in the baseline method, it was not optimized for other operating temperatures before deployment.

We inject faults during the inference accuracy simulation and to each partial sum of the binary layers. A partial sum is generated after activating each group of m_{wl} word lines. Accumulating all the a^s will give the overall activation of a layer.

We have trained a four-layer (256 neurons per layer) densely connected BNN with MNIST and Fashion-MNIST benchmark datasets and a nine-layer CNN with the same VGG topology as [246] for the CIFAR-10 dataset, with the difference that the number of neurons is kept at 256 in the linear layers. We have used a constant learning rate of 6×10^{-3}, the ADAM optimizer with the default setting, and the cross-entropy loss function.

We have quantized the MNIST and Fashion-MNIST datasets to +1 and −1 as a pre-processing step, so the input of all the layers including the first layer's inputs are binary and require one cycle to apply the

input to the crossbar without any degradation in inference accuracy. For CIFAR-10, we have used random horizontal flip and cropping types of data augmentation.

NN with quantized partial sums is trained with Algorithm 5, whereas NN with full precision partial sums is trained with the algorithm from [246]. The trained weights of the linear layers are mapped directly, but the convolution layers are fully unrolled (each kernel is flattened to a column vector) and then mapped to different crossbars with dimension $m \times n$, where m is 2× the row size of the transposed weight matrix of the linear layers and the unrolled weight matrix of the convolution layers. In case the row size of the weight matrix is larger than the row dimension of the crossbar, the weight matrix can be split across multiple crossbars as proposed in [64]. During inference, multiple word-lines are activated concurrently in a single step s and the resulting partial sum current I_{ps} is sensed by the respective sensing technique, for example, the ADC for the un-quantized (a^s) [126] and the 1-bit sense amplifiers for the quantized partial sum (a^s_{quant}) [65]. The resulting sensed value is aggregated to obtain the overall activation of the layer.

For the Monte Carlo simulation to represent different chip instances with process variations, we have performed 100 inference runs for the evaluation of multi-level perceptron (MLP) for each dataset, but we have performed 10 evaluation runs for CNN as convolution operations are computationally expensive. In this paper, we present the mean (μ_{acc}) and standard deviation (σ_{acc}) of the inference accuracy.

11.1.3.2. Pre-deployment Performance and Comparison With Related Works

Here, an ideal reference current is considered for both ADC and 1-bit sense amplifiers to isolate the effect of device non-idealities. We discuss the effect of variations in the reference current circuits in Section 11.1.3.2.

Table 11.2.: Pre-deployment: Depicts change in the training accuracy from the baseline training algorithm to the proposed training algorithm and related works [65] and [121]. Here, ideal accuracy is reported before deployment. Static manufacturing and dynamic thermal variations are not considered. Post-deployment: Comparison of the change in post-mapping inference accuracy from the training accuracy under process (PV) and temperature variations (TV). The column "↔ $\hat{I_{ref}}$" elaborates the inference accuracy of the proposed and related method methods at temperatures below and above 85°C.

Method	Variations	↔ $\hat{I_{ref}}$	MNIST	CIFAR-10
Pre-deployment				
[65]	None		−0.1% (retrain)	−1.1% (retrain)
			−0.4% (mapping)	−78.7% (mapping)
[121]	None		−1.96%	−3.47%
Proposed	**None**		**−0.82%**	**−3.68%**
Post-deployment				
[122]	TV	T ≤ 85°C	−20% → −45% at 37° → 67°C	-
[121]	TV	-	-18.28%	-14.5%
[123]	TV at fixed temp.	T≤ 85°C	-	−8.78% at 25 → 85°
		T> 85°C	-	−23% at 85 → 125°
	TV with temporal temp. change	T<=85 °C	-	−1% at 25 → 55°C
			-	−23% at 55 → 85°C
Proposed	PV + TV	T≤ 85°C	**−0.98% at 25 → 85° C**	**−0.19% at 25 → 85° C**
		T > 85°C	**−3.56% at 85 → 125° C**	**−1.42% at 85 → 125° C**

Pre-deployment section of Table 11.2 compares the performance of the proposed training algorithm (without considering variations) with the related works where the training algorithm is modified and uses similar datasets as ours. Compared to the original BNN [246], our proposed training algorithm achieves a similar inference accuracy with an accuracy difference of only ±0.82, 0.1, and 3.68% for the MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively. The change in inference accuracy is also similar to related works [65, 121]. Furthermore, the training curve of the proposed training algorithm is similar to the original BNN [246] as shown in Fig. 11.7. We found that the MNIST is more sensitive to partial sum quantization in this instance. Therefore, the discrepancy between the baseline training curve and the proposed technique for MNIST is greater than that for Fashion-MNIST. However, the difference can still be considered insignificant. For example, accuracy degradation is 2.74% reported for MNIST in Table 11.3

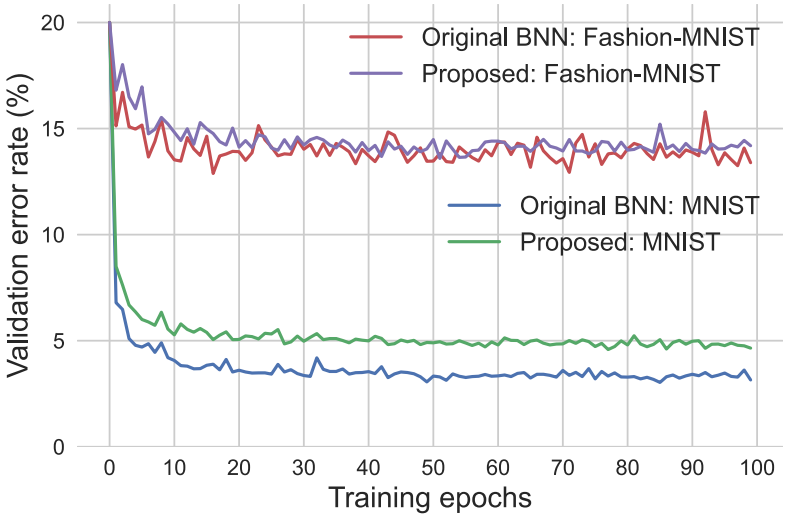


Figure 11.7.: Training curves of an MLP on MNIST and Fashion-MNIST. The training trend of the proposed modified quantization algorithm is similar to the original BNN [246]. The lower the validation error rate, the better the performance. **It is recommended to view this figure in color.**

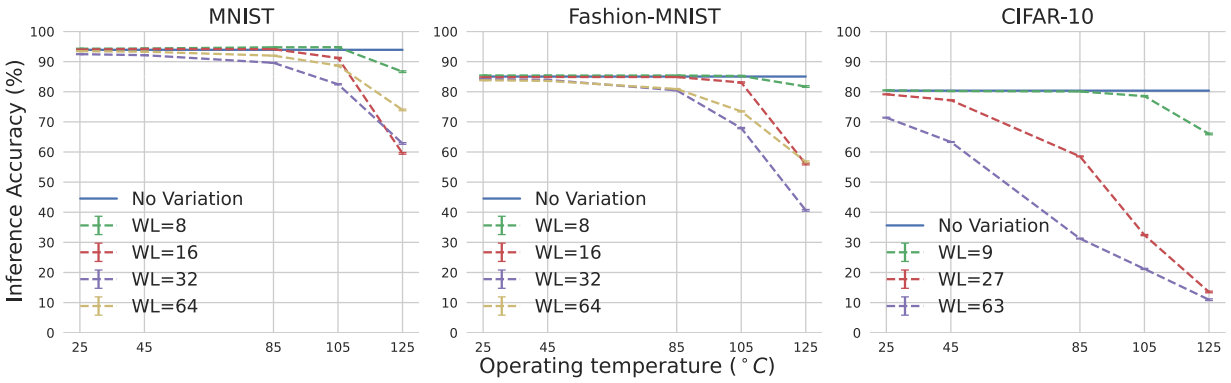


Figure 11.8.: Impact of activating more word-lines concurrently on the inference accuracy of the MNIST, Fashion-MNIST, and CIFAR-10 dataset with both process and temperature variations for baseline I_{ref} with quantized activations. Inference accuracy decreases with more word-lines activations and increasing operating temperature. **It is recommended to view this figure in color.**

but for Fashion-MNIST it is 1.76%. In addition, the overall trend of the training curve for the original BNN and the proposed method is similar. Nevertheless, our proposed training algorithm achieves an inference accuracy comparable to [246] and [65] even at a significantly smaller number of parameters (8× for the linear and 3× for the convolutional layers). Furthermore, when the size of the NN increases, our proposed method achieves a similar accuracy compared to [246], as shown in Table 11.3 for the MNIST and Fashion-MNIST datasets. Similarly, for CIFAR-10, the change in accuracy decreases from 3.68% to 3.06%. In addition, [65] does not quantize the partial sum of the first and last layers, as opposed to our method. In general, the effectiveness of the proposed training Algorithm 5 can be deduced from the pre-deployment inference accuracies of Table 11.2.

Table 11.3.: Difference between inference accuracy of the proposed training algorithm and original BNN as the number of neurons increased for MNIST and Fashion-MNIST datasets. As the width of the model increases, the accuracy difference becomes negligible.

Neurons	MNIST	Fashion-MNIST
256	-2.74%	-1.76%
512	-1.87%	-0.45%
1024	-1.28%	+0.15%
2048	-0.82%	+0.06%

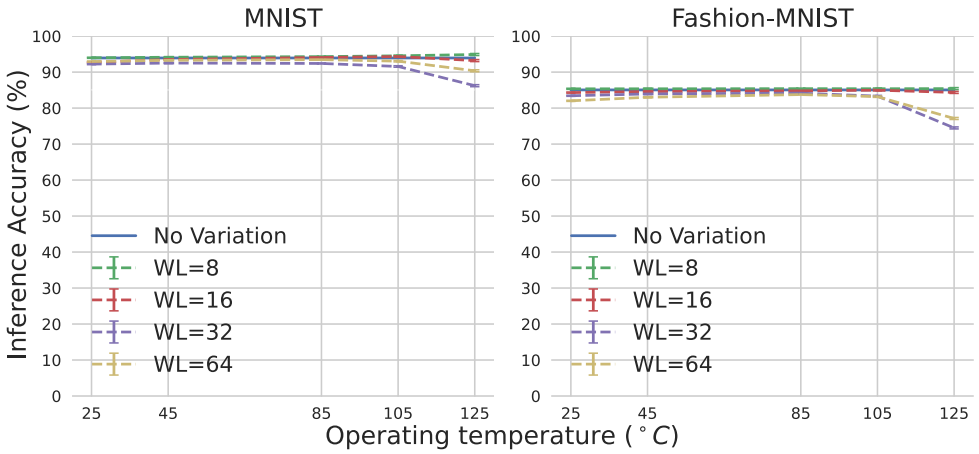


Figure 11.9.: Effect of activating more word-lines concurrently on the inference accuracy of MNIST and Fashion-MNIST dataset with process and temperature variations for the proposed method for generating optimal I_{ref}. Inference accuracy remains stable with more number of word-lines activations and increasing operating temperature. **It is recommended to view this figure in color.**

Process Variation

When only static process variation is considered, the mean inference accuracy of un-quantized partial sum a^s [126] sensed with an ADC decreases to 68.4%, 72.63%, and 42.71% for the MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively. However, the inference accuracy of the quantized partial sum a^s_{quant} [65] sensed with a comparator remains stable, as shown in Fig. 11.6. The CNN model trained for CIFAR-10 is more sensitive to variation compared to the MLP with the same number of word-lines activated. The accuracy of CIFAR-10 decreases by $\approx$ 36% with a^s [126] and $\approx$ 2% with a^s_{quant} [65].

Activating more word-lines concurrently, such as $m_{wl} = 64$, will lead to a higher miss-quantization rate for a^s [126], but it takes a smaller number of total steps $\bar{S}$ to calculate the final activations of a layer, which leads to less overall accumulated errors. Therefore, the inference accuracy is higher when more word-lines are activated. Although a^s [126] can match the inference accuracy of a^s_{quant} [65] at a higher m_{wl}, it will also require a higher precision of the ADC, which is very costly [240]. Also, at higher process variation, inference accuracy will decrease, for example, at a variation of $\approx$ 9.3%, the inference accuracy of Fashion-MNIST decreases to 56.26% and CIFAR-10 decreases to 18.62%.

Although the inference accuracy of a^s_{quant} [65] remains stable under static process variation, it decreases to 73.97%, 56.7%, and 66.91% for the MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively, as the dynamic temperature variation of the device increases as shown in Fig. 11.10. This shows that quantizing a^s alone cannot mitigate temperature variations and is not optimal for the overall operating temperature of the device.

Dynamic Thermal Variations

The inference accuracy of the MNIST, Fashion-MNIST, and CIFAR-10 datasets remains stable up to 85°C, as shown in Fig. 11.10. Therefore, we have chosen the boundary temperature T_b as 85°C. However, note that T_b could be generated at ±2°C from the temperature T_b to achieve a comparable inference accuracy, as Fig. 11.10 shows that the inference accuracy is comparable around the temperature 85°C. The proposed Algorithm 6 was able to find that in one constant step, therefore, it reduces the time complexity of the search step to $O(1)$ (the best possible case) from $O(\bar{H})$ as done in [239].

As a result, with our proposed self-healing approach, that is, calculating $\hat{I_{ref}}$ at $T_b = 85°C$ improves the inference accuracy by up to 16%, 20%, and 12% for the MNIST, Fashion-MNIST, CIFAR-10 datasets, respectively. Note that calculating $\hat{I_{ref}}$ at the highest operating temperature (125°C) will not provide

an acceptable inference accuracy at room temperature, even though it provides better accuracy at high operating temperatures. For example, the inference accuracy of MNIST is reduced to 68%, but the proposed method provides 92.94% inference accuracy.

The post-deployment section of Table 11.2 reports the inference accuracy of the proposed method compared to related works [122, 121, 123] considering both static manufacturing and dynamic thermal variations during the run-time. Compared to related works, our proposed method is significantly robust to both static manufacturing and dynamic thermal variations. For example, in the worst case, the accuracy is degraded by 3.56% for MNIST and 1.42% for the CIFAR-10 dataset. Whereas, accuracy degrades by as much as 23% for related works. To further underscore the robustness of our approach, we have broken down the inference accuracy at $< 85°$C and $> 85°$C operating temperatures since we have generated $\hat{I_{ref}}$ at 85°C. In general, post-deployment inference accuracies highlight how efficient our proposed reference generation approach is for robustness to static manufacturing and dynamic thermal variations.

Although calculating $\hat{I_{ref}}$ at $T_b = 85°C$ improves the overall accuracy, the inference accuracy degrades at high and low temperatures. In the case of MNIST and CIFAR-10, the accuracy degradation is insignificant and outperforms the baseline method for all the simulated temperatures. However, for Fashion-MNIST, the accuracy degradation is slightly higher in comparison. Consequently, at the temperature $25°C$, the inference accuracy of our proposed method is $\sim 1\%$ below the baseline. However, please note that, compared to the entire operating temperature range and all datasets, our approach is significantly robust to both static manufacturing and dynamic thermal variations.

Impact of Number of Concurrent World-lines Activation

When only static process variation is considered, m_{wl} does not affect the inference accuracy with a^s_{quant}, as depicted in Fig. 11.10. However, when both static process variations and dynamic temperature variations are considered, the inference accuracy decreases with more concurrently activated word-lines and increasing temperatures for the baseline (I_{ref} generated at 25°C), as shown in Fig. 11.8. Our proposed method allows activation of up to 64 word-lines concurrently for the MNIST and Fashion-MNIST datasets, as depicted in Fig. 11.9. Also, for the CIFAR-10 dataset, up to 9 word-lines can be activated. Since CIFAR-10 is very sensitive to more word-line activation at higher temperatures (see Fig. 11.8), a comparatively smaller number of word-lines can be activated. We find that the accuracy degrades by more than 10% for CIFAR-10 at higher word-line activation at higher temperatures, e.g., $> 85°$C. Therefore, only a limited number of word-lines can be activated for CIFAR-10.

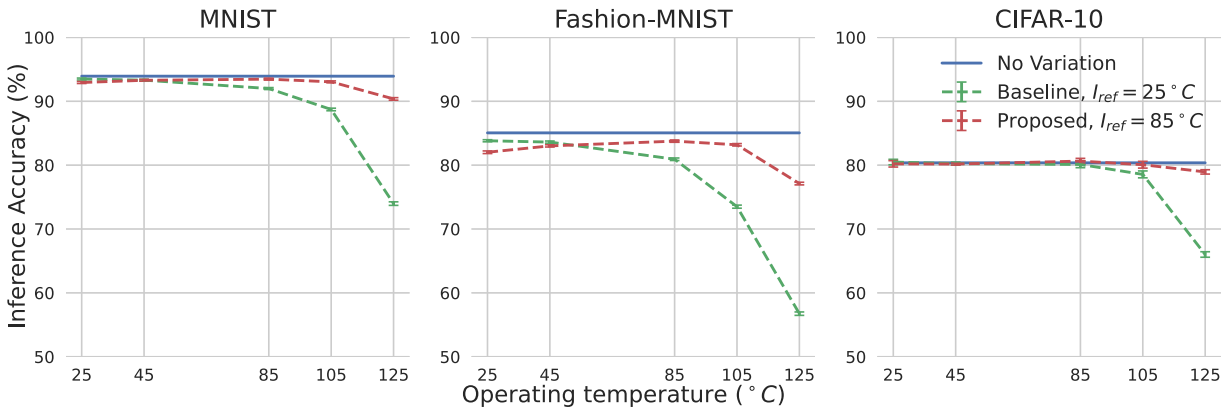


Figure 11.10.: The inference accuracy of a) MNIST, b) Fashion-MNIST, and c) CIFAR-10 datasets under both process and temperature variation. The operating temperature of the device is increased from $25°C$ to $125°C$. The green curve shows, for a reference current generated at $25°C$, temperature-induced shifts in MTJ resistance for the operating temperature from $25°C$ to $125°C$ and the corresponding change in inference accuracy. **It is recommended to view this figure in color.**

Analysis of the Inference Accuracy under $\hat{I_{ref}}$ variation

We consider the variation of $\hat{I_{ref}}$, based on [247] ($\approx$ 2%). There was no noticeable change in the inference accuracy (accuracy change $\leq$ 2%) of the proposed method compared to the accuracy with an ideal $\hat{I_{ref}}$, as shown in Table 11.4 when both process and temperature variations are considered. Therefore, this further underscores the robustness of our approach to variations in both the crossbar array and the reference generation circuits.

Table 11.4.: Analysis of the inference accuracy for Fashion-MNIST under I_{ref} variations when process and temperature variation is also considered.

	Mean Inference Accuracy				
	25°C	45°C	85°C	105°C	125°C
No Variation	85.06%				
Ideal I_{ref}	82.02%	83.012%	83.76%	83.184%	77.13%
$I_{ref} \pm \sim 2\%$	81.69%	82.3%	82.658%	81.46%	75.41%

11.1.4. Scientific Impact of This Work and Contributions

The major contributions and their broad impacts are summarized as follows:

- **Robust Design-Time Reference Current Generation:** We introduced a design-time method for generating reference currents that are robust against both process and temperature variations. Therefore, our approach improves the reliability and accuracy of STT-CiM NN accelerators without additional runtime overhead.
- **Scalability Across Memristor Technologies:** The proposed algorithm-hardware co-design methods are adaptable to other emerging resistive memory technologies, even though the evaluation is performed for STT-MRAM.
- **Implicit Variation Aware NN Training:** We proposed a quantization-aware NN training algorithm using STE to improve the sensing margin of comparator. Improvement in the sensing margin has been shown to be robust to process variation of STT-MRAMs. Therefore, research in the direction of increasing the sensing margin is a viable option for improving robustness against process variation of STT-MRAM.
- **Computational Efficiency:** We propose to integrate the proposed reference current generation into the design phase. Therefore, expensive runtime adaptations such as dynamic remapping or frequent recalibrations is eliminated, significantly reducing the computational overhead and energy consumption.
- **Deployment of CiM Architectures in Dynamic Environments:** The ability of the proposed method to handle different operational temperatures and process variations makes it a viable option for the deployment of NN in STT-CiM-based AI accelerators in an uncertain environment. Our approach ensures that CiM-based AI accelerators can adapt to environmental changes without manual intervention, which is crucial for edge computing applications.

11.1.5. Section Summary

In summary, this section has analyzed the impact of device-to-device variations and runtime temperature fluctuations on the inference accuracy of BNNs mapped onto STT-MRAM-based crossbars. The baseline inference accuracy degraded significantly due to the process and temperature variations of the MTJ devices. We proposed a training algorithm and a hardware co-design technique to mitigate this degradation of the inference accuracy of the STT-CiM NN accelerators. Specifically, our proposed method finds the

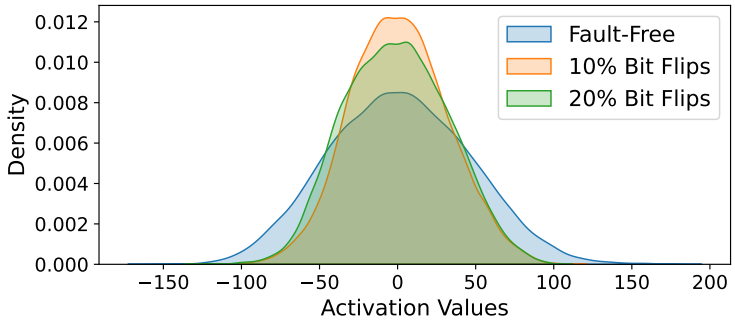


Figure 11.11.: Change in activation distribution due to faults. **It is recommended to view this figure in color.**

optimal reference current for the entire operating temperature range of the device during the design time. In contrast to the existing works, our proposed method does not require any per-chip or runtime adaptation to deal with process and temperature variations. Thus, it eliminates the need for costly per-chip re-training, runtime re-mapping, circuitry for temperature sensing, STT bit-cell modification, or network reconstruction. Consequently, our proposed technique improves the inference accuracy by up to 20.51%. In addition, our proposed training algorithm can quantize each partial sum without needing any extra intermediate neurons. It enables higher parallel computation and is significantly more robust against both process and temperature variations.

11.2. Self-Healing Bayesian NNs

BayNNs are a natural fit for memristor-based CiM architectures as they offer to reduce some of their inherent costs. Additionally, one can leverage NVM non-idealities such as resistance variation and stochastic switching for efficient BayNN inference computations [84].

Although memristor-based CiM offers several advantages for BayNN implementations, mitigating the reliability challenges associated with these devices is of utmost importance. To reiterate, non-idealities such as manufacturing and runtime variations, defects, and failures can significantly reduce the accuracy of inference [128, 144]. In safety-critical applications, it is essential not only to give uncertainty in prediction but also to maintain high accuracy even in the presence of these non-idealities. The primary focus of existing works is either uncertainty estimation or fault tolerance. Therefore, either fault tolerance or uncertainty estimation aspect is overlooked. Furthermore, they sometimes require expensive fault modeling, variation injection, and NAS.

In this section, our aim is to close this gap by offering *inherently self-immune* BayNN that 1. does not require any implicit non-idealities modeling, 2. is generalizable across different NVM technologies and non-idealities 3. easier to train, 4. CiM implementation friendly, and 5. is still able to provide uncertainty estimates without reducing its accuracy. These properties are of critical importance to ensure the reliable deployment of BayNNs implemented with CiM in safety-critical applications.

This section is based on our conference and journal papers IEEE DATE24 [145].

11.2.1. Problem Statement

In existing work [128, 30], it has been shown that due to the non-idealities of the NVMs, the distribution of weighted sum shifts from the trained distribution. Figure 11.11 shows the distribution change due to 10% and 20% bit-flip faults. Based on this observation, it can be stated that *non-idealities of NVMs* adds additive or multiplicative noise to the weighted sum of a layer. Existing work [75] also supported this statement based on their NVM variation model.

Therefore, we hypothesize that adding stochastic additive and multiplicative components to the weighted sum of a layer would increase the robustness of an NN against these types of noise. This is because such stochasticity at the weighted sum of a layer adds implicit additive and multiplicative noise, and the model learns to be robust to such noise during the Bayesian inference.

11.2.2. Methodology

In this section, we introduce *inverted normalization and affine Dropout* for a self-healing BayNN. In our approach, the order of the affine transformation in the normalization layer is reversed, with the affine parameters randomly dropped. Furthermore, as mentioned in [144], the scaling factors (γ) of the normalization layers amplify the drift of the parameters and, in turn, reduce the accuracy of the network. Thus, treating them as random parameters can potentially improve robustness.

In addition, in our approach, normalization is done for each output instance or group of neurons in a layer. This adds another layer of robustness by standardizing the weighted sum of each layer in the case of distribution shifts (see Fig. 11.11) due to non-idealities. This approach has been proven to be effective in improving robustness in works [30] and [205, 204].

11.2.2.1. Inverted Normalization

To reiterate 11.2.2, normalization techniques such as batch normalization or layer normalization adjust the input to zero mean and unit variance in different dimensions. Subsequently, an *optional* affine transformation is carried out using learnable parameters. The main aim was to give the NN the freedom to reverse the normalization if it is beneficial. However, in practice, it is unlikely that the affine parameters will learn to reverse the normalization rather than participate in the optimization process. We have observed the distribution before and after the affine transformation of the normalization layers of several topologies and found that they are different.

Based on this observation, we propose the inverted normalization layer. In our approach, we treat the affine parameters of the normalization as parameters similar to the weights and biases of the NN. That is, their learning objective only is to minimize the loss using the gradient descent algorithm. For simplicity, in the following, the affine parameters γ and β are referred to as weights and biases of the inverted normalization layer.

Also, unlike traditional normalization methods, the affine transformation in our approach is *necessary* and is performed before normalization. Since normalization is still performed on the transformed input, the learning process remains stable. The supporting results are shown in Section 11.2.3. Figure 11.12 shows the flow diagram for the conventional and proposed inverted normalization layers.

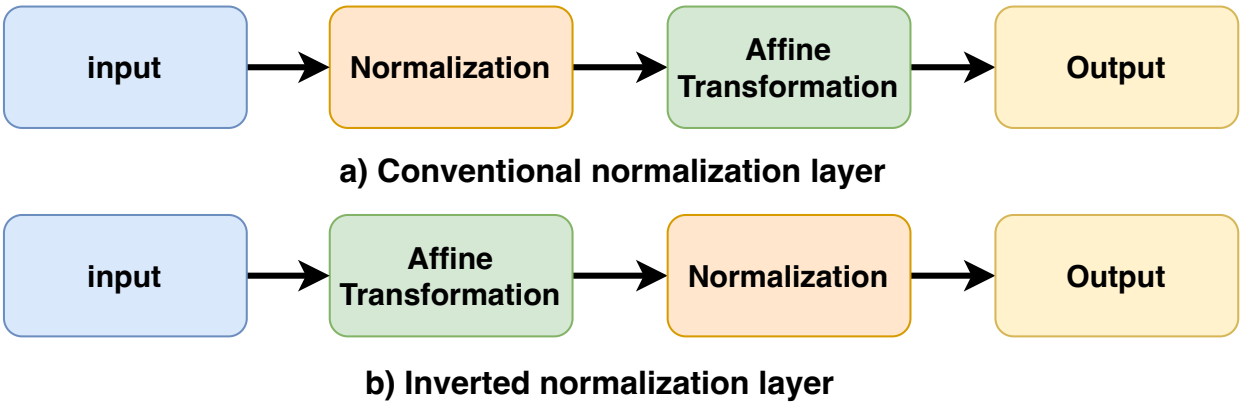


Figure 11.12.: Computation flow of the proposed and conventional normalization layers.

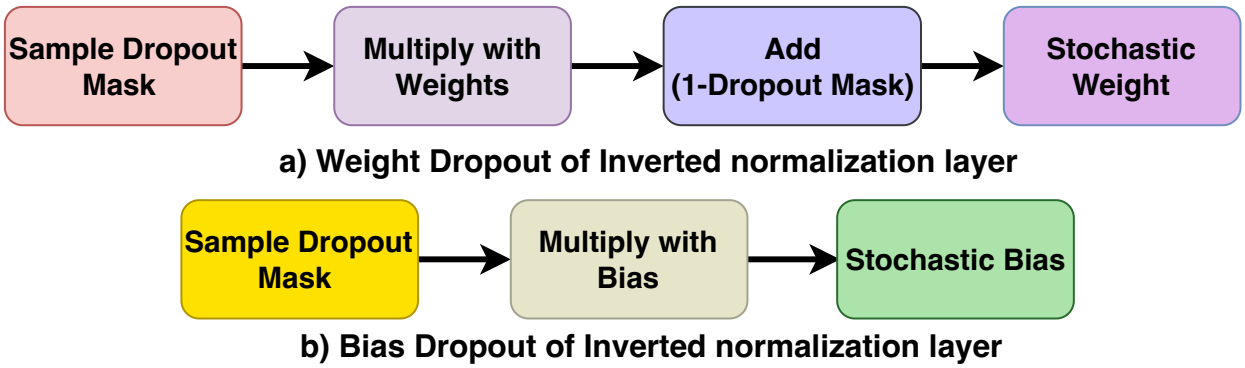


Figure 11.13.: Operation flow for the proposed affine parameters (weight and biases) Dropout.

11.2.2.2. Affine Dropout

To add stochasticity to the weighted sum, we randomly drop the weight and bias of the inverted normalization layer independently with probability p. Unlike traditional Dropout techniques, the weight and bias of the inverted layer normalization are dropped to *ones* and *zeros*, respectively. Since the weights of the inverted normalization layer multiply the weighted sum, it cannot be dropped to zero. Dropping the weights to ones and zeros effectively ignores the weights and biases.

To implement the proposed affine Dropout, two binary ($[0, 1]$) Dropout masks are sampled from the Bernoulli distribution with Dropout probability p. Subsequently, the masks are multiplied by the weights and biases of the inverted normalization layer. This will set the dropped weights and biases to zero. Lastly, a (1-Dropout mask) is added to the masked weights to ensure that the dropped weights are one. The flow diagram of the proposed affine Dropout is shown in Fig. 11.13. Subsequently, the affine transformation and normalization are performed as shown in Fig. 11.12 (b).

11.2.2.3. Proposed Dropout Implementation

Individual components of the weights and biases can be dropped independently, referred to as the element-wise Dropout. Alternatively, the entire weight and bias vector can be dropped at the same time, referred to as the vector-wise Dropout. Vector-wise Dropout is more efficient as it only needs to sample one Dropout mask for the entire weights or bias vector, irrespective of their length. In addition, it only needs *one* random number generator per layer to implement Dropout in the CiM architecture. Therefore, in this work, we employ vector-wise Dropout. Reusing the Dropout module among all layers, number of Dropout module can be reduced to only one for the entire model.

11.2.2.4. Initialization of Affine Parameters

Initialization of weights and biases is important to achieve not only comparable accuracy but also to allow the proper learning of the affine parameters. Traditional normalization methods initialize γ and β as ones and zeros, respectively. However, our inverted normalization initializes its weight and biases randomly. Otherwise, the initial weights and biases can produce the same gradient and update in the same way throughout the training. In addition, random initialization allows for more randomness in the weighted sum, which can potentially improve robustness.

Specifically, the weights are initialized from a normal distribution with a mean and variance of 1 and σ_γ, $\mathcal{N}(1, \sigma_\gamma)$. Initially, this either scales up or down the weighted sum randomly. Similarly, biases are sampled from a normal distribution $\mathcal{N}(0, \sigma_\beta)$. Alternatively, weights and biases can be initialized from uniform

distributions $\bar{\mathcal{U}}(0, k_\gamma)$ and $\bar{\mathcal{U}}(-K_\beta, K_\beta)$, respectively. Where K_β and k_γ are positive integers that define the range of uniform distribution.

11.2.2.5. Bayesian Inference and Uncertainty Estimation

In the Bayesian paradigm, every weight in the network is modeled as a probability distribution, which allows us to capture model uncertainty. As shown by Gal. et al. [35], a NN trained with conventional Dropout and weight decay (L2 regularization) is an approximation of a Gaussian process. During inference, multiple forward passes through the network, each time sampling different Dropout masks, will give a stochastic output distribution. The average of all of these outputs gives the final prediction.

Our proposed affine Dropout acts as an alternative to the conventional Dropout in a Bayesian setting. Multiple forward passes can be made through the network, with each time independently sampling weight and bias masks for each layer, giving an output distribution. The final prediction is obtained from the average of those outputs. Uncertainty in prediction can be obtained from the variance of outputs or by calculating the NLL for a classification task.

By integrating Bayesian inference with our inverted normalization and affine Dropout techniques, we present a model that is not only robust to various forms of noise and non-idealities, but is also capable of quantifying the uncertainty associated with its predictions. This makes it particularly suitable for deployment in safety-critical applications where both performance and reliability are crucial.

11.2.3. Results

11.2.3.1. Simulation Setup

Evaluated Models and Training Settings To show generalizability, we have evaluated our method on four different deep-learning tasks: image classification, audio classification, autoregressive time series forecast, and semantic segmentation. For image classification, we used the CIFAR-10 benchmark dataset on the ResNet-18 2D-convolutional NN (CNN) topology. The Google speech command dataset is evaluated on a five-layer 1D-CNN model, referred to as M5, for the audio classification task. In addition, an NN with two LSTM layers and a classifier layer was used for the time-series forecast. Lastly, the DRIVE (digital retinal images for vessel extraction) dataset is trained on the popular U-Net topology for biomedical semantic segmentation tasks.

In terms of bit-precision, ResNet-18 is binarized using the algorithm [47]. Semantic segmentation is a much harder task, thus activations of the U-net model are quantized to 4 bits using [173], but their weights are binarized. On the other hand, the LSTM and M5 models are quantized to 8 bits to show the generalizability of our approach in a range of bit precision.

The proposed inverted normalization layer is applied following all the convolutional layers as a drop-in replacement for conventional normalization layers. A Dropout probability of 0.3 is used for all our models. In U-Net, we have normalized across groups of $\frac{C_{out}}{8}$ channels together with the same train-time and test-time behavior as Group Normalization. Here, C_{out} refers to the output channels. In other models, we have normalized each input instance, the same as the Layer Normalization method.

NVM Non-idealities Model In this work, we have abstracted the circuit-level details into an algorithmic model in the Pytorch framework. We modeled both the manufacturing and thermal conductance variation as an additive and multiplication noise, as suggested by [75]. Additive variation is modeled as $\mathcal{N}(0, \sigma)$ and multiplicative variation is modeled as $\mathcal{N}(0, \sigma)$. As in the works [75, 143], those noises are injected into weights for 8-bit weights, but in the case of binary NNs, they are injected into normalized activations before applying the Sign(.) function.

Other post-manufacturing and infield non-idealities of NVMs, such as programming errors, and retention faults, are modeled as bit-flips. For binary and quantized parameters, the random bits are flipped in each simulation run. We also conducted a different experiment on the LSTM model, where we injected (random) uniform noise with varying strength.

We perform 100 Monte Carlo fault simulation runs, simulating 100 chip instances, for each variation and bit-flip scenario, and report the mean and standard deviation of accuracy. **It is recommended to view this figure in color.**

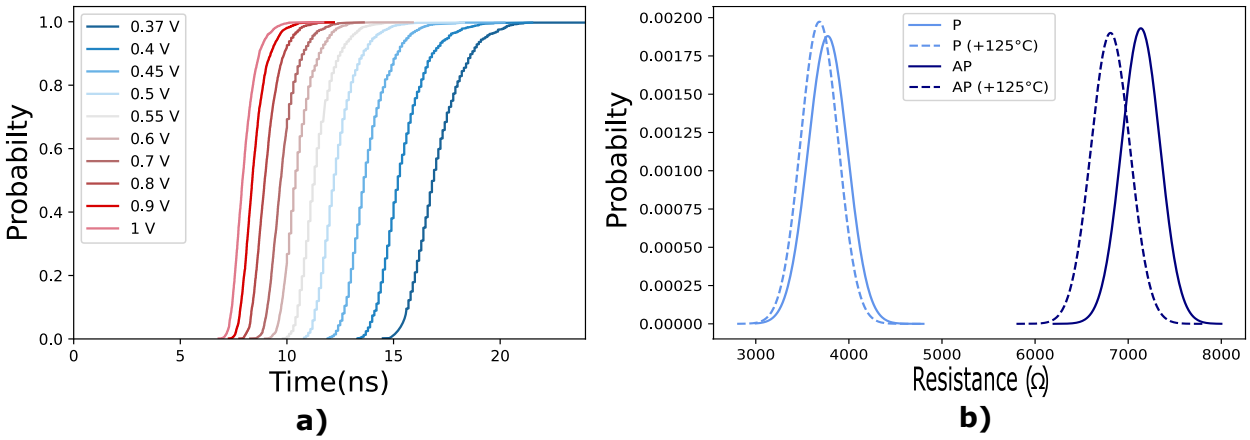


Figure 11.14.: Examples of non-idealities: (a) Stochastic switching in magnetic memories under different voltages and (b) influence of temperature on the resistance distributions (Monte Carlo simulations).

11.2.3.2. Baseline Inference Accuracy

Although our aim is to improve robustness against the non-idealities of the NVMs, it is equally important to achieve an accuracy comparable to the baseline NN and SOTA Dropout-based bayNNs [85, 86]. As shown in Table 11.5, the inference accuracy of our method is comparable to the baseline on all the datasets and topologies evaluated. In the worst case, the accuracy of CIFAR-10 is 0.66% below the SpatialSpinDrop [86] method. However, using a smaller Dropout probability, such as 0.1 or 0.2 can improve accuracy, but may be less robust to NVM non-idealities. However, the proposed approach outperforms the conventional NN in all metrics.

Table 11.5.: Summary of inference accuracy of the proposed method and related works evaluated on different datasets, bit-precision, metrics, and topologies. Here, W/A refers to the bit-precision of weights and activation.

Topology	Dataset	metrics	W/A	NN	SpinDrop	SpatialSpinDrop	**Proposed**
ResNet-18	CIFAR-10	Accuracy ↑	1/1	89.01%	89.82%	90.5%	**89.82%**
M5	Google Speech Commands	Accuracy ↑	8/8	83.97%	84.83%	-	**85.28%**
U-Net	DRIVE	mIoU ↑	1/4	66.87%	67.93%	64.6%	**67.54%**
LSTM	Atmospheric CO2	RMSE ↓	8/8	0.1264	0.1534	-	**0.1219**

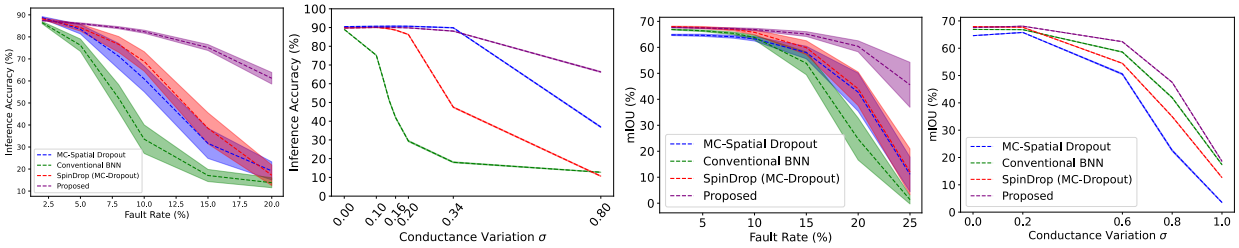
11.2.3.3. Analysis of Variation Robustness

We only compare our work with related Dropout-based BayNNs targeting CiM architecture. Variational inference-based works [49, 104] represent completely different learning and network topologies. Therefore, they are ignored for comparison.

In terms of robustness to NVM conductance variations, our proposed approach is robust to additive and multiplicative variations on all datasets with varying degrees of variation. As shown in Figs. 11.15 and

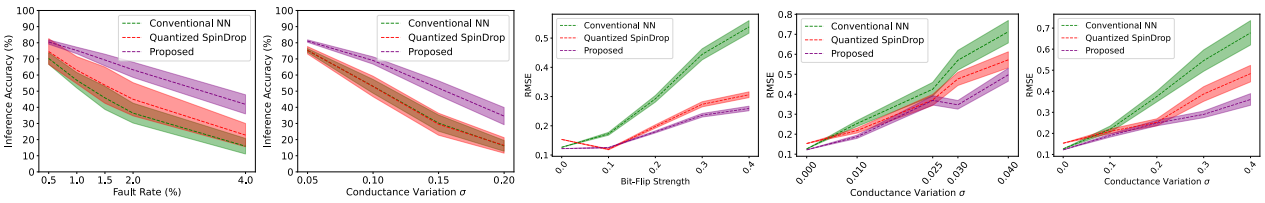
11.16, our approach leads to a graceful degradation in accuracy or other metrics evaluated. Specifically, our approach improves inference accuracy by up to 55.62% compared to other Dropout-based BayNN approaches and 53.55% compared to conventional NN. In the case of semantic segmentation, there is a marginal improvement in accuracy. In LSTM-based time series prediction, the RMSE score is reduced by up to 30.2% and 46.7% for additive and multiplicative variations, respectively.

We have performed a detailed evaluation of additive variations, but only in the LSTM model was an evaluation of multiplicative variations performed. Nevertheless, our results translate to multiplicative variations on other datasets.



(a) CIFAR-10 dataset on ResNet-18 **(b)** DRIVE dataset on U-Net

Figure 11.15.: Evaluation of robustness of ResNet-18 and U-Net topologies on CIFAR-10 and DRIVE datasets. The shaded region shows ± one standard deviation variation from the mean. The left and right figures of both datasets illustrate the evaluation of bit-flips and additive conductance variations, respectively. **It is recommended to view this figure in color.**



(a) Googe speech commands dataset on M5 **(b)** Atmospheric CO2 forecast with LSTM-based autoregressive model

Figure 11.16.: Evaluation of conductance and bit-flip robustness of ResNet-18 and U-Net topologies on CIFAR-10 and DRIVE datasets. The shaded region shows ± one standard deviation variation from the mean. Here, the first and second figures of both datasets show the evaluation of bit-flips and additive conductance variations, respectively. In (b), the last figure shows multiplicative conductance variations. **It is recommended to view this figure in color.**

11.2.3.4. Analysis of Bit-flip faults Robustness

Similarly to conductance variations, our method shows significant robustness to bit-flip faults in all datasets. Our approach can improve accuracy by up to 42.06% compared to other Dropout-based BayNN approaches and 58.11% compared to conventional NN. In addition, the standard deviation in accuracy is smaller for our approach compared to other approaches, as shown by the narrower band in Figs. 11.15 and 11.16. In the LSTM model, our approach reduces the RSME score by up to 51.84%.

11.2.3.5. Uncertainty Estimation

To reiterate, typically, it is assumed that the training data and test data are derived from the same distribution (ID). However, when these distributions are not aligned, such as when the test images are corrupted, e.g., via rotation, or contaminated with measurement noise, the model's predictive uncertainty should increase, indicating that the model's prediction is dubious.

To evaluate this feature of the proposed BayNN, we conducted two identical experiments to [49] to investigate the effect of shifting the dataset in various ways. In the first experiment, images were gradually rotated in 7-degree increments in 12 stages. In the second scenario, escalating random uniform noise levels

were introduced. As depicted in Fig. 11.17, as a consequence of these shifts in the data distribution, the accuracy of the inference decreases, and the NLL score increases. We use NLL as an uncertainty estimation metric, similar to the work [49]. In general, a lower NLL score is desired for ID data and a higher NLL score for OOD data.

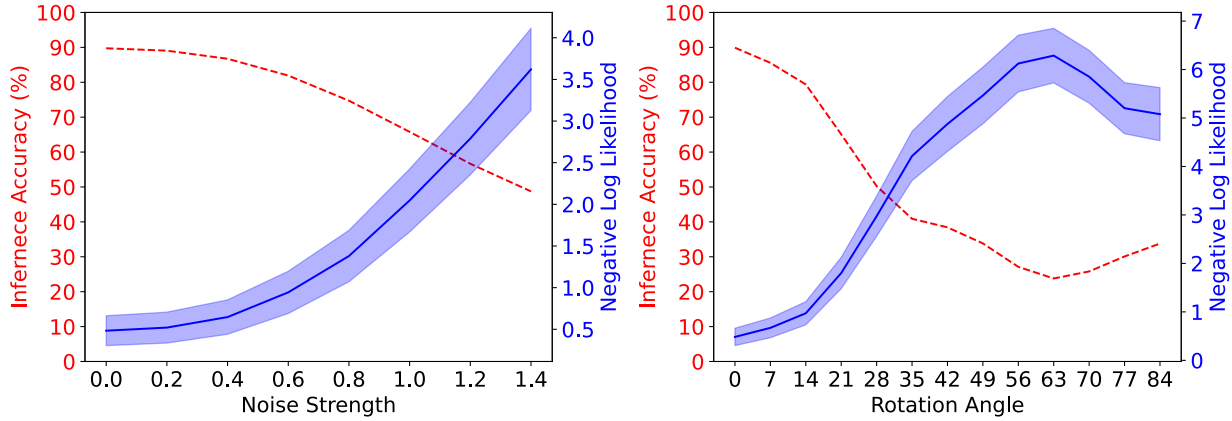


Figure 11.17.: Evaluation of the proposed method on OOD data. (Left) Uniform noise is added to images and (right) images are rotated to shift the distribution. **It is recommended to view this figure in color.**

The NLL score can be utilised to detect OOD data. A value greater than a predetermined threshold, such as the average NLL score on the test dataset, indicates the presence of OOD data. Using this methodology, we can identify up to 55.03% and 78.95% of OOD instances for uniform and random rotation experiments, an improvement of 14.61% compared to [49].

11.2.3.6. Impact of Initialization on Inference Accuracy

As mentioned, initialization of the weights and biases of the proposed inverted normalization is important to improve not only the accuracy of fault-free inference but also the robustness against NVM non-idealities.

The weights and biases of the proposed inverted normalization are initialized from normal distributions with a value of 0.3 for σ_γ and σ_β. We have found that initializing with larger σ_γ and σ_β can improve robustness to variations and bit-flip faults, as it introduces more randomness to the weighted sum. However, it can reduce the accuracy of the baseline by 1-2%.

11.2.4. Scientific Impact of This Work and Contributions

This work builds on the knowledge and insight gained from our previous work on vector dropout [48] in designing and implementing the proposed Dropout. Specifically, this work reinforces that research in the direction of vector Dropout and Dropout layer design, considering the information flow, is an effective method for BayNN implementation.

Other scientific impacts of this work are summarized as follows:

- **Randomness in the Weighted Sum:** The showed that by introducing randomness into the weighted sum of a layer fault-tolerance can improve fault-tolerance. Therefore, research in the direction of non-zero randomness is an intriguing option for fault tolerance.
- **Scalability to Other Noise Model:** In the case of memristor-based CiM architecture, non-idealities introduce additive and multiplicative noise to the weighted sum. Therefore, our aim was to introduce this kind of randomness. Our approach can be adopted for other kinds of noise models of other AI accelerators, e.g., by modifying affine transformations.

- **Fault Tolerance In BayNN:** The work motivates the need for fault tolerance in addition to providing uncertainty estimates in safety-critical AI applications. Existing works ignore one aspect or the other. Research in this direction is important to improve the overall reliability of safety-critical edge AI applications.
- **Implicit Noise Injection:** This work showed that implicit noise is highly attractive for scalable (different memristor technologies, non-ideality types, NN tasks, topologies, and datasets) fault tolerance. Whereas, explicit noise injection may not be scalable as it is usually targeted for specific use cases. In our approach, we introduce implicit noise via the proposed inverted normalization and stochastic affine transformation. However, the general concept can be adopted via other methods.
- **Applicability:** The applicability of any approach is important when it comes to research adaptations. Our proposed approach is easy to implement as it replaces only the traditional normalization and dropout layers with the proposed stochastic normalization layer. Therefore, it can be considered a plug-and-play method, which is highly attractive.

11.2.5. Section Conclusion

In this section, we present a self-immuned Bayesian neural network method for reliable CiM implementation in safety-critical applications. We introduce an inverted normalization layer that performs the affine transformation first, then normalization. In addition, we propose the affine dropout, which introduces randomness and, in turn, introduces implicit noise into the weighted sum of each layer where it is applied. Consequently, the combined effect of those leads to inherent robustness to NVM non-idealities in their IMC implementation. We show that the fault-free prediction performance is comparable to that of SOTA BayNN and conventional NN with various parameter precisions and deep learning tasks to show scalability. Furthermore, in various tasks and non-ideality scenarios, our approach can improve inference accuracy by up to 58.11% and RMSE by up to 51.84%. Nevertheless, our approach does not compromise the uncertainty estimation capabilities of BayNN, with up to 78.95% detection of OOD instances.

12. Runtime Periodic Maintenance Approaches

Although self-healing approaches offer fault tolerance without system intervention, runtime maintenance approaches, in contrast, can maintain accuracy close to the baseline and even offer guaranteed error correction. Runtime maintenance approaches considered in this thesis involve active runtime interventions to manage faults and uncertainties in edge AI accelerators. The interventions can be scheduled periodically or triggered by online testing mechanisms that detect anomalies, performance drops, or when the accuracy is below a predefined threshold. Runtime maintenance ensures that the system can promptly address and correct issues as they arise, maintaining optimal performance and reliability.

This chapter presents several runtime periodic maintenance approaches that are significantly low-cost, such as run-time re-calibration, and virtually zero overhead methods, such as approximate scrubbing and zero overhead head ECC.

12.1. Runtime Re-Calibration For Fault-Tolerance

Due to immature and non-deterministic manufacturing processes of memristors, runtime variations, and faults such as thermal fluctuations and retention faults, the activation distributions of neurons can shift from their trained distribution [128].

During training, BatchNorm normalizes activation distributions, and during inference, trained statistics are employed. Existing works have proposed re-calibrating trained statistics parameters of normalization layers in the presence of variations [128]. However, such re-calibration incurs high hardware overhead as the re-calibration needs to be performed for each layer of NN, a large number of calibration inputs is required, and large buffer memory is required to store intermediate results for the re-calibration.

In this section, we present an efficient re-calibration method that allows post-mapping and in-field re-calibration with significantly reduced overhead compared to existing solutions. We introduce approximate batch normalization (ApproxBN), which approximates running mean and variance computation to reduce re-calibration complexity and requires constant memory irrespective of batch size. ApproxBN encapsulates and completely removes batch normalization during hardware inference. We also introduce a functional ATPG approach for compacting re-calibration inputs. The overall effect minimizes computing, memory, and parameters needed for re-calibration and inference. Our method improves manufacturing yields, the in-field chip failure rate, and, consequently, inference accuracy.

This section is based on our publications IEEE VTS22 [204] and IEEE D&T [205].

12.1.1. Problem Statement and Motivation

To reiterate, the batch normalization layer standardizes each activation of NN to approximately zero mean and unit variance. In BatchNorm, the element-wise mean ($\mu = \frac{1}{B} \times \sum_{b=0}^{B} \mathbf{z}_i$) and variance ($\sigma^2 = \frac{1}{B} \times \sum_{b=0}^{B} (\mathbf{z}_i - \mu)^2$) is precisely calculated for a mini-batch size of B.

Such a precise calculation has $O(k)$ memory overhead (assuming that a two-pass algorithm is used for numerical stability, which first computes mini-batch μ then σ^2) in hardware and is computationally expensive as it requires B iterative addition, subtraction, and square operation.

Our goal is to bring the activation distribution approximately closer to the fault-free (ideal) distribution with low overhead.

In neuromorphic hardware, only one input can typically be applied at a time. Hence, the test inputs for the re-calibration and inference are applied serially. In addition, training data (inputs) can range from a few thousand to roughly over a million [8]. Therefore, storing the images on-device and re-calibrating is infeasible.

12.1.2. Approximate Batch Normalization (ApproxBN)

12.1.2.1. During Training and Post-mapping Re-calibration

We propose the approximate batch normalization method, which approximately calculates the running mean ($\bar{\mu}$) and variance ($\bar{\sigma}^2$) of a mini-batch of size B with the formula proposed by work in [248]. The estimation uses the median (Me), *approximate batch normalization*, which approximates the running mean ($\bar{\mu}$) and variance ($\bar{\sigma}^2$) of a mini-batch of size k. We adopt the method proposed in [248] for the approximation, but our method is not intended as a standalone mean and variance estimation technique as in [248]. Our method is integrated into a larger re-calibration framework for efficient re-calibration of NNs. The estimation employs the median (Me), minimum (l), and maximum (h) values of a mini-batch to calculate $\bar{\mu}$ and $\bar{\sigma}^2$:

$$\bar{\mu} = \begin{cases} \frac{l+Me+h}{4} & \text{if } K < 25 \\ M & \text{if } K \geq 25 \end{cases} \tag{12.1}$$

$$\bar{\sigma}^2 = \begin{cases} \frac{(l-2Me+h)^2}{36} + \frac{(h-l)^2}{12} & \text{if } K < 25 \\ \frac{h-l}{4} & \text{if } 15 < K < 70 \\ \frac{h-l}{6} & \text{if } K > 70. \end{cases} \tag{12.2}$$

For larger batch sizes, estimating μ and σ becomes simpler. However, smaller mini-batches result in lower overhead and more re-calibration steps.

The proposed modifications have constant O(1) memory overhead, irrespective of the mini-batch size, as it requires only three variable storage. In addition, the computation required to calculate the running mean and variance is much simpler and requires fewer numbers of addition, subtraction, and division operations.

A comparator can be utilized for the calculation of the minimum and maximum value in hardware. Also, binary NN has predefined possible activations, e.g., when WL word-lines are activated concurrently for a crossbar of row size m, and possible activation values are $-\varpi, \ldots, \varpi$. Where, ϖ is $\frac{m}{WL}$. Hence, the unique activation list does not need to be stored and sorted for the median calculation. Instead, a counter c is used to count the number of unique activations of each neuron of NN for a mini-batch, and a 32-bit integer ψ is utilized to keep track of the unique activation values for a mini-batch. Each bit position of ψ represents whether a certain activation is present or absent, i.e., the most significant bit of ψ represents the ϖ activation value. During re-calibration, a bit position of ψ is set when an intermediate activation is present, and the median value is evaluated by finding the activation that corresponds to the $\lceil \frac{c}{2} \rceil$-th set position of ψ after activating all the word lines. After each re-calibration step, ψ is reset. Therefore, the proposed modification has the memory overhead of only one variable for the median calculation in hardware.

12.1.2.2. During Inference

To reiterate, during inference, the population mean and variance are used. However, batch normalization still requires the storage of two variables ($\bar{\mu}$ and $\bar{\sigma}^2$) and the learned parameters (γ and β) per neuron. Therefore, it has a $4C_{out}$ memory requirement for inference, where, C_{out} is the number of neurons of a linear layer and output channel of convolutional layers. In addition, after each proposed re-calibration step, those variables need to be written in hardware. Furthermore, batch normalization calculations have some power and latency overhead during inference, even with approximate batch normalization, since batch normalization is performed per neuron.

Consequently, we propose a method that removes not only the storage requirements but also the batch normalization calculation during inference by reducing all the variables and learned parameters of batch normalization to a bias (b'). It is based on the fact that the original batch normalization equation 2.5 can be mathematically simplified as:

$$\begin{aligned} \boldsymbol{y} &= \frac{\mathbf{z} - \bar{\boldsymbol{\mu}}}{\sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}} \times \boldsymbol{\gamma} + \boldsymbol{\beta} \\ \boldsymbol{y} \times \sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon} &= (\mathbf{z} - \bar{\boldsymbol{\mu}}) \times \boldsymbol{\gamma} + \boldsymbol{\beta} \times \sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon} \\ \boldsymbol{y} \times \frac{\sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}}{\boldsymbol{\gamma}} &= \mathbf{z} - \bar{\boldsymbol{\mu}} + \frac{\boldsymbol{\beta} \times \sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}}{\boldsymbol{\gamma}}. \end{aligned} \tag{12.3}$$

Since all of the variables and parameters of batch normalization are constant during inference, they can be squashed to a constant $\bar{\boldsymbol{\alpha}}$ such as

$$\boldsymbol{y} \times \frac{\sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}}{\boldsymbol{\gamma}} = \mathbf{z} + \bar{\boldsymbol{\alpha}}. \tag{12.4}$$

where, $\bar{\boldsymbol{\alpha}}$ represents $\frac{\boldsymbol{\beta} \times \sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}}{\boldsymbol{\gamma}} - \bar{\boldsymbol{\mu}}$. Since BNN binarizes each intermediate activation after batch normalization using the sign(x), the constant term $\frac{\sqrt{\bar{\boldsymbol{\sigma}}^2 + \epsilon}}{\boldsymbol{\gamma}}$ can be removed as it only scales the numerical value of the activation and is not expected to change the sign of the activation, i.e., $-1 \rightarrow 1$. Therefore, the batch normalization can be further reduced to

$$\boldsymbol{y} \approx \mathbf{z} + \bar{\boldsymbol{\alpha}}. \tag{12.5}$$

The α term can be combined with the original bias $\boldsymbol{b}$ to give $\boldsymbol{b'} = \boldsymbol{b} + \bar{\boldsymbol{\alpha}}$, which replaces the bias in equation 2.2.

12.1.3. Post-Manufacturing Functional ATPG

We propose a post-manufacturing functional ATPG method that selects a small subset of the validation dataset as re-calibration inputs for the post-manufacturing re-calibration instead of selecting them randomly as done in [30]. Since the NN is not trained on the validation dataset, the performance of NN on validation datasets is close to that of the testing dataset. As a result, we hypothesize that the activation shift due to non-idealities of the NVM memories on validation datasets will be close to that of testing datasets. Hence, test inputs from the validation dataset are a better candidate for the re-calibration instead of training datasets as used by work in [30] and [128].

In our approach, we initially select one random test input per class, ensuring each class is represented during re-calibration and maintaining dataset balance. An imbalanced re-calibration dataset could result in biased mean and variance estimates towards the dominant class, negatively impacting re-calibration.

Our objective is to track distribution changes caused by hardware non-idealities by identifying similar inputs, ensuring that the majority of activation shifts result from non-idealities. Various similarity measurement methods exist, with L1 distance, which calculates the sum of the absolute differences between

two vectors, being popular. We sample (without replacement) the most similar inputs to random inputs per class in each step based on L1 distance. The dataset selection process ends after $\frac{\text{size of the total re-calibration data}}{\text{number of classes in the dataset}}$ sampling steps to ensure adequate data coverage. The combined effect of our functional ATPG significantly reduces re-calibration test inputs compared to [30] and [128], as demonstrated later in Table 12.2.

Algorithm 7 Post-manufacturing re-calibration with approximate batch normalization and functional ATPG algorithm. Here, $\mathcal{D}$ is the dataset, $\mathbf{x}_{\text{class}}$ is the per-class random test input, D_{class} is all the inputs of a class, Ω is the momentum, and $\mathbf{x}$ is the ATPG generated test inputs.

▷ 1. Functional ATPG
for class = 1 to classes **do**
▷ 1.1 select one random test input per class
$\mathbf{x}_{\text{class}} \leftarrow rand(\mathcal{D})$
▷ 1.2 calculate L1 distance i.e., similarity score
$d \leftarrow \sum |\mathbf{x}_{\text{class}} - D_{\text{class}}|$
▷ 1.2 select inputs with lowest L1 distance, i.e., most similar test inputs
$\mathbf{x} \leftarrow\subseteq sort(d)$
end for
▷ 2. post manufacturing re-calibration
$\Omega \leftarrow 0.9$ ▷ set default momentum value
for mini-batch re-calibration input $\mathbf{x}$ in $\mathbf{x}$ **do**
for L in binary layer with batch normalization **do**
▷ 2.1 calculate approximate mini-batch $\bar{\mu}$ and $\bar{\sigma}^2$
$\bar{\mu} = \frac{l+M+h}{4}$
$\bar{\sigma}^2 = \frac{(l-2M+h)^2}{36} + \frac{(h-l)^2}{12}$
▷ 2.2 Update running mean and variance
$\mu_c \leftarrow (1-\Omega) \times \mu_C + \Omega \times \bar{\mu}$
$\sigma_c^2 \leftarrow (1-\Omega) \times \sigma_C^2 + \Omega \times \bar{\sigma}^2$
end for
end for

12.1.4. Overall Re-Calibration Workflow

Initially, our proposed ATPG method samples a re-calibration dataset stored in hardware. During post-mapping re-calibration, mean and variance estimates are calculated using Formulas 12.1 and 12.2 for each mini-batch. The running means and variances of the batch normalization layer are then updated. This process continues until all batches are re-calibrated. In-field re-calibrations are done periodically in the CiM architecture during the in-field operation.

The proposed post-mapping and in-field re-calibration, considered in hardware to account for device-to-device variation non-idealities and their effects, necessitate individual device re-calibration.

12.1.5. Simulation Results

12.1.5.1. Simulation Setup and Fault Injection Framework

In this work, we focus on permanent defects that are modeled as stuck-at faults and manufacturing variation in MTJ-based crossbars, but the proposed methods can be applied to other emerging NVM-based crossbars. Multiple stuck-at faults are assumed to occur on the crossbar.

The variation of MTJ-based cells depends on the operating temperature and device characteristics, while the stuck-at faults depend on manufacturing processes [69, 82]. The device properties and operating

temperature for the simulation are summarized in Table 12.1. The stuck-at-fault model is based on [31], and the faults are injected into the weight matrix of all layers at random locations by setting them to either LRS or HRS. Due to the complementary bitcell design of XNOR cells [64], faults are injected into the active word lines only, as injecting faults into the inactive cells has no impact on the overall current sum. However, for the manufacturing variation, we have built a custom PyTorch-based simulation framework to evaluate the inference accuracy of the MTJ under per-crossbar and device-to-device manufacturing variation. At first, Monte-Carlo spice simulations based on statistical models of MTJs are performed to generate the conductance distribution of LRS and HRS states of MTJs. Then, a random conductance value for each cell is drawn from their distribution and converted to per-cell current with the knowledge of bit-line voltage. The per-cell currents are accumulated to obtain the partial current sum (I_{ps}) for the number of activated word-lines. A linear ADC compares the I_{ps} with I_{ref} to inject a fault into each partial activation of binary layers, which is accumulated to get the overall activation of a layer after activating all the word-lines. The overall analytical fault model and fault injection flow are shown in Figs. 12.1 (a) and (b). Due to manufacturing variation, I_{ref} of ADC is also subject to variation, but their variation is not considered in this paper to isolate the effect of device non-idealities only.

We have trained a four-layer binary MLP (256 neurons per hidden layer) on MNIST and Fashion-MNIST and a nine-layer CNN with the same topology as [246] on the CIFAR-10 datasets, with the difference that the number of neurons is kept at 256 in the linear layers. We have used the ADAM optimization algorithm with the default setting in Pytorch, a constant learning rate of 6×10^{-3}, and a cross-entropy loss function. We have used 20% of training data as validation data without training NNs on them. The model that achieved the best accuracy on the validation dataset is used for simulation.

Table 12.1.: MTJ parameters and simulation setup

Parameter	Value	Parameter	Value
VDD	0.8V	MTJ radius	20nm
Nominal Temperature	27°C	Ra	$7.5\Omega\mu m^2$
CMOS Library	Globalfoundries 22FDX	TMR @ 0V	220%
Free/Oxide layer thickness	1.3/1.48 nm	'AP'/'P' resistance	19 kΩ/6 kΩ

Both MLP and CNN are trained with the original BNN Algorithm [246]. The trained weights of the linear layers mapped to a crossbar with a dimension (rows × columns) of 512×256, and the fully unrolled weight matrix of the convolution layers are split across multiple different crossbars of the same dimension as the linear layers. We have used weight matrix splitting and current accumulation as proposed in [64].

We have simulated our method on 100 crossbars networks for both CNN and MLP with all the benchmark datasets, to simulate the effect of per-chip variations and defects. The manufacturing variation is increased up to 13.95% ($3\sigma_{\text{MRAM}}$), and the stuck-at-fault rate is increased up to 10% of the overall weights of linear layers and fully unrolled convolutional layers to evaluate the impact of different fault scenarios. The proposed post-manufacturing re-calibration is done for the proposed method according to Algorithm 7, but no re-calibration is done for the baseline model. Here, σ_{MRAM} represents manufacturing variations of STT-MRAM cell.

12.1.6. Analysis of the per-device re-calibration with BatchNorm

An optimal momentum value for BatchNorm is crucial. Values too low or high, e.g., 0.1 or 0.9, may decrease mean accuracy to $30-40\%$ from the reported accuracies. The best results are obtained with a value between 0.2 and 0.3.

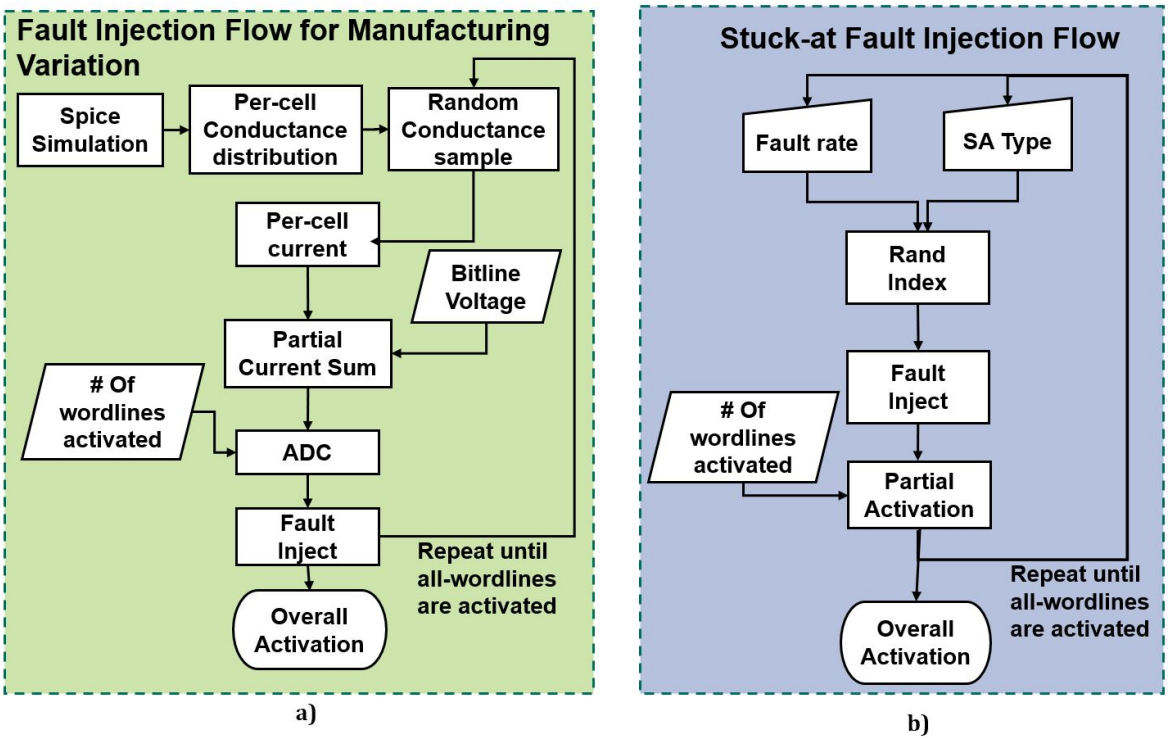


Figure 12.1.: Overview of analytical fault modeling for a) manufacturing variation and b) stuck-at faults. Faults are injected into intermediate activation for manufacturing variation and weights for stuck-at faults.

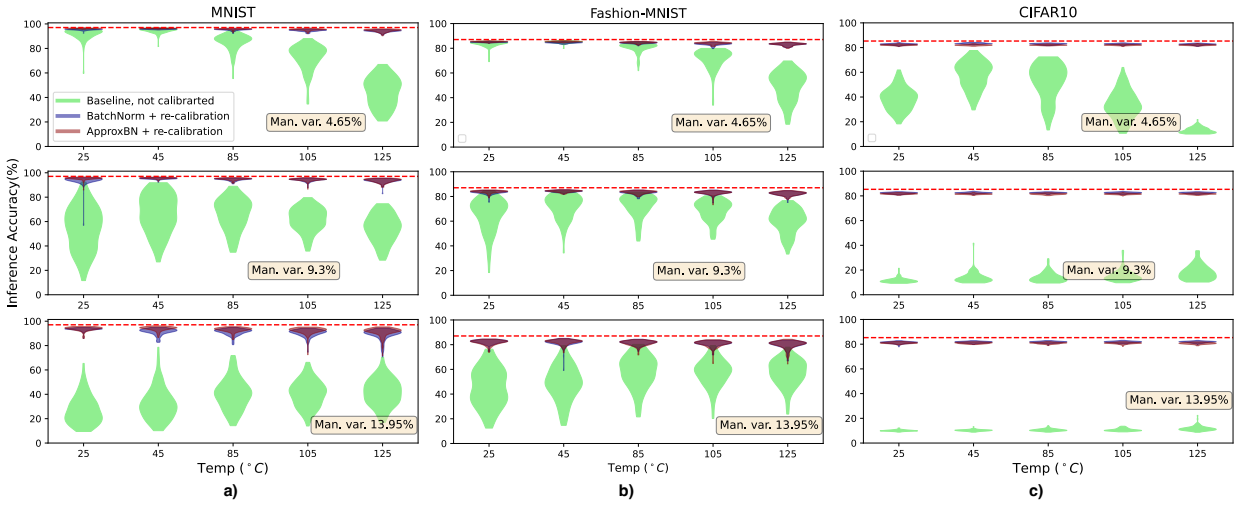


Figure 12.2.: Impact of thermal variations on the inference accuracy of the MNIST, Fashion-MNIST, and CIFAR-10 datasets when runtime temperature increases from 25 to 125°C. Also presented is the influence of various manufacturing variations (man. var.). The horizontal lines show the training (ideal) inference accuracy. **It is recommended to view this figure in color.**

12.1.6.1. Manufacturing Variation

As the variation increased, the mean accuracy of the baseline decreased to 45.09%, 27.54%, and 10.07% for MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively. Fig. 12.2 shows the overall inference accuracy distributions at room temperature (25°C) with violin plots (first plot in each graph).

However, re-calibrating BatchNorm with 100 – 200 ATPG-generated test input increases the worst-case mean inference accuracy to 93.41%, 82.71%, and 81.52% for the MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively. Re-calibrated accuracy is within 3 – 4% of the training accuracy. Furthermore, the violin plots in Fig. 12.2 exhibit low variation, indicating the robustness of our approach. For each ~ 4 – 5%

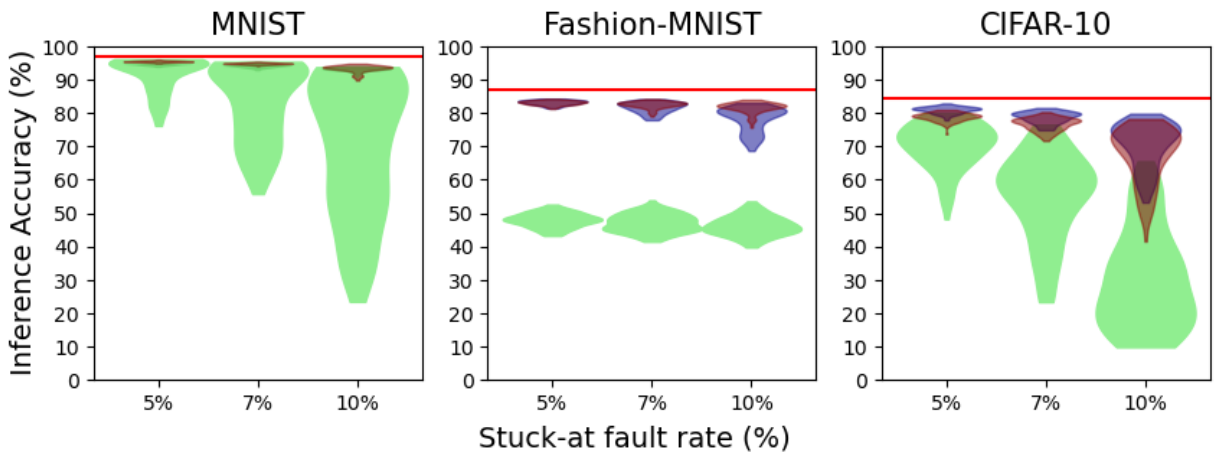


Figure 12.3.: Impact of stuck-at faults on the inference accuracy of the MNIST, Fashion-MNIST on MLP and CIFAR-10 datasets. The horizontal lines show the baseline (ideal) inference accuracy. **It is recommended to view this figure in color.**

increase in NVM cell variation, 50 additional ATPG inputs are required to prevent up to 10% degradation in accuracy. Consequently, the proposed method improves the manufacturing yield up to 100% (97% in the worst case), assuming that a chip fails when post-mapping accuracy drops below 80% of the training accuracy.

12.1.6.2. Thermal Variations

As the temperature increases from room temperature, baseline inference accuracy drops to 32.04%, 48.79%, and 10.24% for MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively, as shown in Fig. 12.2.

However, re-calibrating BatchNorm with 300 ATPG generated inputs achieves a worst-case mean inference accuracy of 94.04%, 83.08%, and 82.60% for the MNIST, Fashion-MNIST, and CIFAR-10 datasets at the 4.65% manufacturing variation. This brings the accuracy within ~3% of the ideal training accuracy. At high manufacturing and thermal variations, the worst-case mean inference accuracy is reduced by up to 4.97%. Moreover, the variance in accuracy from the mean is significantly low, as demonstrated by the flattened shape of the violin plots. Consequently, the proposed method has up to 100% lower in-field chip failure rate.

The CIFAR-10 dataset on CNN is highly sensitive to thermal variations and requires all 300 re-calibration inputs. On the contrary, MLPs with MNIST and Fashion-MNIST require more inputs only at higher temperatures. Similarly to manufacturing variations, the number of re-calibration inputs needs to be increased as thermal variations increase.

12.1.6.3. Stuck-at-faults

For stuck-at-faults, as the fault rate increased from 5% to 10%, the mean baseline inference accuracy decreased to 69.26%, 46.2%, and 30.3% for MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively, as shown in the Fig. 12.3. Post-mapping re-calibration improved worst-case mean inference to 93.35%, 78.13%, and 70.56% for the same datasets, respectively. Furthermore, the median, first, and third quartiles of the proposed method are much closer to the baseline accuracy (as shown in Fig. 12.3). Ultimately, the proposed method can improve the yield by up to 100% (61% in the worst case) under the same assumptions.

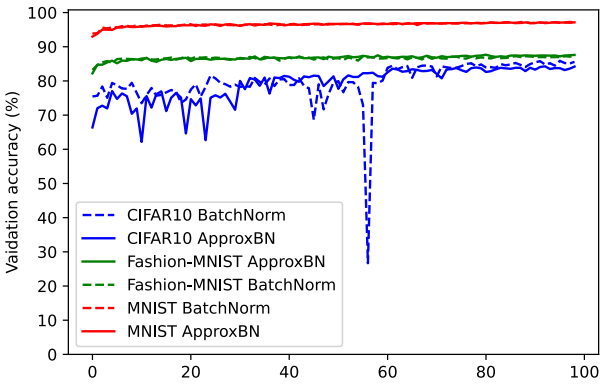


Figure 12.4.: Training curve for the validation accuracy of ApproxBN compared to BatchNorm on different datasets. **It is recommended to view this figure in color.**

12.1.7. Analysis of the per-device re-calibration with ApproxBN

Similar to BatchNorm, optimizing the momentum is critical. Otherwise, a drastic degradation in accuracy can be observed. We found that a mid-range momentum value (0.5 to 0.6) is optimal for ApproxBN.

12.1.7.1. Training

Training BNN with ApproxBN from random initialization achieves similar performance, as shown in Fig. 12.4. Consequently, inference accuracy of NN with ApproxBN is within +0.06%, +0.4%, and −1.59% of NN with BatchNorm for MNIST, Fashion-MNIST, and CIFAR-10, respectively. ApproxBN can also be initialized from a pre-trained BNN with BatchNorm.

When using ApproxBN for re-calibration, a similar mini-batch size to the training mini-batch size is necessary to prevent a significant drop in mean inference accuracy (≤10%). Therefore, training with smaller mini-batches is advantageous.

12.1.7.2. Manufacturing Variation

ApproxBN reduces the number of ATPG-generated re-calibration inputs to 100 for up to 13.95% ($3\sigma_{\text{MRAM}}$) manufacturing variation. Compared to BatchNorm, it requires up to 50% fewer test inputs and 90% fewer re-calibration mini-batches (see Table 12.3). Nevertheless, the accuracy distribution of ApproxBN overlaps with BatchNorm for other datasets, as shown in Fig. 12.2. Furthermore, ApproxBN has a yield improvement similar to BatchNorm, with a slightly better worst-case yield of 99%.

It is worth noting that the best-case scenario for BatchNorm is identical to the ApproxBN approach. Thus, in terms of overhead, approxBN and BatchNorm can be similar in some cases, such as with low manufacturing variations. However, this may not hold true for higher manufacturing variations, even if the number of mini-batches and their sizes are merged, as BatchNorm requires more re-calibration inputs when manufacturing variation increases.

12.1.7.3. Thermal Variations

In-field re-calibration with ApproxBN requires only 150 ATPG-generated test inputs, resulting in 50% fewer re-calibration inputs and 77% fewer mini-batches than BatchNorm. Fig. 12.2 shows violin plots for ApproxBN similar to BatchNorm across all datasets, thermal, and manufacturing variations.

Table 12.2.: Comparison of the proposed method with the related work in terms of inference accuracy, buffer memory overhead, number of re-calibration steps performed, and re-calibration test inputs. CNN is benchmarked on CIFAR-10, and the same mini-batch size and calibration test inputs as [30] are assumed for [128].

		Proposed	[30]	[128]
Test inputs (% of training data)	MLP	25(0.04%)	-	-
	CNN	100(0.2%)	2600(5%)	100%
# of mini-batches	MLP	1	-	-
	CNN	5	13	13
Memory overhead (per neuron)		12 Bytes	800 Bytes	800 Byte
Inference accuracy	CNN	80.95%	83.6%	83.57%

Table 12.3.: Comparison of re-calibration batch size, number of mini-batches applied, and test inputs for the BatchNorm and proposed ApproxBN. The number in the brackets represents in-field re-calibration settings.

	ApproxBN (Proposed)	BatchNorm
Test inputs	100 (150)	100-200 (300)
# of mini-batch	4 (5)	20-40 (15)
Mini-batch size	25 (30)	5 (20)

In MNIST, ApproxBN has a slightly flattened inference accuracy distribution due to lower variations compared to BatchNorm. However, for other datasets, the accuracy distribution overlaps with BatchNorm (see Fig. 12.2). Consequently, it achieves up to 100% lower in-field chip failure rate.

12.1.7.4. Stuck-at-faults

Similarly, inference accuracy distributions of ApproxBN under stuck-at faults overlap that of BatchNorm for Fashion-MNIST and CIFAR-10 datasets. Achieving a similar median, first, and third quartiles. However, for MNIST, the inference accuracy distribution is slightly lower ($1-2\%$), as demonstrated in Fig. 12.3. Additionally, ApproxBN has the same yield improvement as BatchNorm.

12.1.8. Batch Normalization Parameter Collapsing

ApproxBN does not require batch normalization computation as well as parameters and variables storage for inference, but there is a negligible inference accuracy difference of -0.3%, -1.87%, and $+0.18\%$ for the MNIST, Fashion-MNIST and CIFAR-10 datasets, respectively, as shown in Table 12.4. Furthermore, the memory and computation required for the inference of NN can be reduced significantly as no parameters and variables are required to be stored and no calculations need to be performed for the batch normalization in hardware. In terms of memory, ApproxBN reduces the memory requirement by 12.448 KiloBytes and 37.024 KiloBytes, respectively, for the MLP and CNN architectures used in this paper, assuming that each parameter is stored in a 32-bit floating-point value. Additionally, it reduces calculations performed on the hardware by 5.446 kFLOPS for MLP and 16.198 kFLOPS for CNN, assuming that each operation takes one instruction in hardware.

12.1.9. Analysis of partial re-calibration

We have calibrated each layer BNN separately but injected faults into all the layers to find the most important layers for the re-calibration. We have found that in CNN, re-calibrating only the convolution layers and in MLP, re-calibrating only the first $2-3$ hidden layers is enough to achieve a mean accuracy

Table 12.4.: The effect of removing batch normalization parameters and computation during inference in hardware. Evaluated on testing datasets and without any fault injection.

	Compute BN during inference	MNIST	Fashion-MNIST	CIFAR-10
ApproxBN	Yes	96.39%	85.6 %	83.88%
	No	96.09%	83.73%	84.06%

Table 12.5.: Comparison of mean inference accuracy of partial re-calibration with full re-calibration on 13.95% ($3\sigma_{\text{MRAM}}$) variation.

	MNIST	Fashion-MNIST	CIFAR-10
Full calibration	92.22%	81.6%	80.95%
Partial calibration	92.15%	81.87%	80.31%
# of layers calibrated	3\4	2\4	6\9

$0.07 - 1\%$ lower than full re-calibration, as shown in Table 12.5 when evaluated on 13.95% ($3\sigma_{\text{MRAM}}$) variation, as at that variation rate the inference accuracy degrades the most for the baseline. Therefore, partial re-calibration reduces the number of layers that need to be calibrated by 33.33% for CNN and $25 - 50\%$ for MLP.

12.1.10. Scientific Impact of This Work

The scientific impact of this work and the main contributions can be summarized as follows:

- **Online Re-calibration for Variation-tolerance in BNN:** We show that online re-calibration can improve manufacturing and thermal variation tolerance for binary neural networks. By recalculating the statistics of the batch normalization layers post-mapping and post-deployment, our approach enhances the robustness of BNNs against shifts in activation distributions due to manufacturing and online thermal variations, thereby improving overall inference accuracy.
- **Re-calibration for Retention Fault-tolerance:** We show that the proposed re-calibration method is effective in mitigating retention faults in CiM architectures for the first time. Our approach ensures that the NNs retain their performance over time despite the inherent retention faults, which can otherwise lead to significant accuracy degradation.
- **Partial Re-calibration for Fault-tolerance:** We introduce the partial re-calibration strategy that selectively re-calibrates the sub-set of the layers of a neural network. With partial re-calibration, our method reduces the computational and memory overhead associated with the re-calibration while still maintaining high fault tolerance and accuracy levels.
- **Approximate Normalization:** We introduce the approximate batch normalization method, which reduces the overhead of precise normalization calculations during re-calibration. Specifically, ApproxBN uses simplified statistics to approximate the running mean and variance calculation, significantly lowering the memory and computational requirements without compromising the effectiveness of re-calibration. Furthermore, it allows for the complete removal of normalization calculation during inference, significantly improving the inference efficiency.
- **Scalability:** Our approach, although demonstrated for STT-MRAM-based CiM architectures, is scalable to other memristor technology and edge AI accelerator architectures. Furthermore, our approach is scalable to any NN task, topology, and dataset that uses batch normalization layers.

12.1.11. Section Conclusion

In this section, we have introduced functional ATPG, which selects only up to 0.2% of training data as the re-calibration test inputs. Also, proposed approximate batch normalization, which during re-calibration

approximately calculates the mean and variance and, during inference, completely removes the batch normalization layer. Our proposed technique can improve inference accuracy by up to 72.32%, bringing it within 2.4% of training accuracy, with up to 13.95% (3σ) manufacturing variation and 10% stuck-at faults rate. Compared to existing studies, our proposed technique achieves comparable baseline fault-free inference accuracy on significantly lower overhead.

12.2. Maintaining Retention Faults and Aging Induce Drifts

12.2.1. Problem Statement

The direction of the uni-directional state change of memristive devices is technology-dependent. Assuming MTJ-based crossbars are used, the state changes from $+1 \rightarrow -1$ are far more common than retention faults in the other direction, $-1 \rightarrow +1$. Consequently, these uni-directional retention faults accumulate over time and severely impact the inference accuracy after a certain period, as shown in Fig. 12.7.

12.2.2. Methodology

12.2.2.1. Core Idea

To mitigate retention faults, we propose an *approximate scrubbing technique* to prevent the accumulation of uni- directional faults. The main idea behind this approach is to define a *scrub region SA* and a *non-scrubbing region nSA* where most unstable and stable weights are stored, respectively. The scrub region can then be periodically scrubbed to restore the respective values. In contrast, since the weights in the non-scrubbing region nSA mainly contain stable weights, they are not scrubbed. As a result, the overall cost and latency associated with our approach are further reduced.

We propose two ways to define the scrubbing area: 1) predefined and 2) learnable. In the pre-defined scrub area approach, a pre-specified region of the crossbar is chosen as the scrub area before training the NN model, and defining this region requires a hyperparameter (that defines the scrub area) search. The advantage of this approach is that it has negligible memory overhead, but the drawback is that it does not give the NN the freedom to choose the optimum scrub area for the task, which can lead to comparatively low accuracy. To address this, we introduced the learnable scrub area approach. This approach gives the NN the freedom to adjust the scrub area during training, producing optimal accuracy for the task, given some constraints that are discussed later. However, in comparison to the predefined scrub area approach, it has a slightly higher memory overhead.

During the online operation of the NN, the scrub area need to be frequently scrubbed to restore their respective values. The frequency of scrubbing (scrubbing rate) is important. This is because a too-low scrubbing rate can lead to incorrect prediction between the scrubbing window, and a too-high scrubbing rate can lead to energy consumption due to unnecessary scrubbing. Therefore, we propose to optimize the scrubbing rate based on the memristor technology used, device parameters, environmental factors such as operating temperature since the retention fault rate depends on those factors, scrubbing technique, e.g., hybrid scrub (see Section 12.2.2.5). For example, less frequent scrubbing is performed for ReRAM-based crossbars due to their limited endurance and gradual state change. In addition, the type of scrub can be chosen based on the complexity of the NN task and the size of the NN topology. For instance, for a relatively easier task, a smaller NN topology scrub area can be pre-defined as it has negligible overhead. The scrub area can be learned during training for a harder NN task or bigger NN topology.

To perform scrubbing, a scrub controller is implemented that receives information about the scrub region's shape (irrespective of scrub area definition method), scrub frequency, and scrubbing technique. It performs a write operation on each of the memristor cells in the scrub region (row/column-wise) to restore the respective weight value (+1). As a result, the accumulation of errors can be prevented, and the degradation

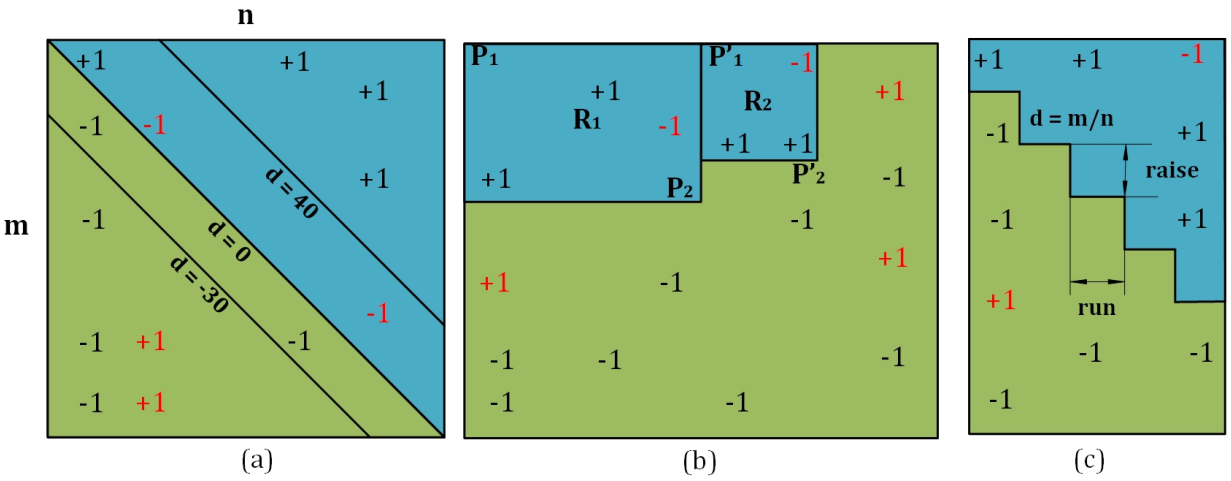


Figure 12.5.: (a) A Crossbar ($m = n$) showing different possible scrub and non-scrub areas with different diagonals d, (b) A Crossbar ($m < n$) showing rectangular shaped (R_1 and R_2) scrub area. Each scrub area requires storing two points ($\mathbf{P}_1$ and $\mathbf{P}_2$), (c) A Crossbar ($m > n$) showing staircase shaped (*raise* and *run*) scrub area for Conv layers of CNN.

of the inference accuracy can be mitigated either completely or partially. However, the scrub controller does not modify the state of the memristor cells in the non-scrubbing region nSA as they are not considered stable.

12.2.2.2. Scalability Challenges

There are some key challenges associated with our proposed technique. One challenge is to define the shape of the scrub region, depending on the difficulty of the NN tasks and NN topology, while maintaining low scrubbing costs. In particular, the scrubbing cost should be low enough to make it scalable for resource-constrained devices. Each scrub area requires the storage of information about the size of the scrub region SA. To avoid significant memory overhead, the space complexity of the definition of SA should ideally be kept constant for a pre-defined scrub area, i.e., $O(1)$. However, for a learnable scrub area, a more complex scrub area definition is preferred, which gives the learning algorithm more freedom to adjust the scrub area and achieve high accuracy. Therefore, the scrubbing overhead is relatively higher. However, ideally, overhead should be relatively lower compared to the size of the weight matrix or NN topology.

12.2.2.3. Proposed Scrubbing Technique

As described before, our objective is to divide the crossbar array into scrub area SA and non-scrub area nSA for both pre-defined and learnable scrub areas, which contain most of the unstable +1 and stable −1 weights, respectively, while keeping the cost of scrubbing to a minimum. Hence, different shapes can define the scrub areas. A few examples of pre-defined scrub area descriptions with constant overhead $O(1)$ can be seen in Fig. 12.5(a) and (b). In a diagonal scrubbing region, a diagonal d is defined in the crossbar array, which separates the scrub area SA from the nonscrub area nSA as shown in Fig. 12.5(a). The value of d is considered an additional hyperparameter for NN training. Alternatively, the learnable scrub area creates clusters of scrub and non-scrub areas that dynamically change during training. It produces a much more complex scrub area by trading off storage overhead and has no hyperparameter but only learnable parameters.

Since the definition of the triangle-shaped scrub region depends only on a single integer d, the storage overhead is constant $O(1)$. The top-right area above d thus defines the scrub area SA, while the bottom-left defines nSA. Increasing or decreasing d can be visualized as the diagonal moving towards the top-right or bottom-left, as depicted in Fig. 12.5(a). The diagonal-shaped scrubbing region leads to more unique

neurons compared to the rectangular scrubbing region depicted in Fig. 12.5 (b). Here, a unique neuron refers to a neuron whose weight vector is different from the weight vector of another neuron in the same layer. Although the number of rectangles can be increased to increase unique neurons, but it will require more storage. Note that storage overhead for the rectangular shape scrubbing still remains $O(1)$ when the number of rectangles increases.

In most CNN topologies, the number of output channels c_{out} increases with the depth of the topology. As a result, the flattened weight matrix has a shape of rows (m) >> columns (n). The simpler diagonal-shaped scrub region will lead to a significantly lower number of unique filters and will degrade the initial (t_0) inference accuracy. Similar to a unique neuron, a unique filter refers to the filter that is not replicated in the weight matrix of a layer at the same exact location, e.g., when $d = 40$ in Fig. 12.5(a), then the area below the diagonal will have the same filters replicated across all c_{out}.

Hence, we propose a more flexible staircase-shaped scrub area that is pre-defined for a comparatively simpler NN task or smaller CNN topology, as shown in Fig. 12.5(c) for the Conv layers of CNNs. The shape can be defined by choosing d_1 = raise and d_2 = run, and has a $O(1)$ storage overhead. The training hyperparameter and storage overhead can be reduced by choosing raise = run. Please note that when run > 1, neurons (output channels) are repeated. Repeated neurons refer to the fact that when $run > 1$, e.g., $run = 2$, then the weights of both neurons in that region are the same.

Furthermore, for a difficult NN task or a larger CNN topology, the scrub area can be learned during training. Therefore, the shape of the scrub area is not needed to be pre-defined, instead, the overhead of the scrub area should be defined, as discussed in Section 12.2.2.4. Consequently, this allows selecting appropriate scrub-areas based on the type of layers (Conv or fully-connected), the shape of layers, the size of NN topology, the difficulty of NN tasks, and where they appear in the network, e.g„ the type of the subsequent layers.

12.2.2.4. Proposed Training Technique

Due to the employed scrubbing, we prefer solutions with +1 and −1 in the respective regions (e.g., +1 in the scrub area and −1 in the non-scrubbing area), as these entries will be less affected by retention faults. As mentioned previously, we propose two types of scrub area definition: 1) pre-defined, and 2) learnable scrub area. The training procedure for each technique is different. This section describes the training procedure for each technique.

Pre-defined Scrub Area In NN training, the loss function expresses the preference for solutions. For the pre-defined scrub area, a penalty function augmenting the original training objective (loss function) can be designed to express our scrubbing preference. In this section, this method is referred to as ①.

As NN training is most commonly described as a minimization problem, the penalty function should exhibit its minimum value at the most preferred configuration, i.e., where all +1 and −1 weights are in the correct region. Consequently, the more weights are placed outside the respective region, the higher the penalty. Note that many of such penalty functions can be designed to satisfy this property. For example

$$\text{Penalty}_1(\mathbf{W}) = \frac{1}{m \times n} \sum_{w \in \text{SA}} (1 - w) + \frac{1}{m \times n} \sum_{w \in \text{nSA}} (1 + w), \tag{12.6}$$

and

$$\text{Penalty}_2(\mathbf{W}) = \frac{1}{m \times n} \sum_{w \in \text{SA}} (1 - w)^2 + \frac{1}{m \times n} \sum_{w \in \text{nSA}} (1 + w)^2. \tag{12.7}$$

The penalty function (12.6) and (12.7) penalizes based on weight value. Therefore, we propose

$$\text{Penalty}_3(\mathbf{W}) = \frac{1}{m \times n} \sum_{w \in \text{SA}} (\text{ReLU}(-w))^2 + \frac{1}{m \times n} \sum_{w \in \text{nSA}} (\text{ReLU}(w))^2. \tag{12.8}$$

which penalizes based on the sign of the weight value. As a result, it allows the learning algorithm more degrees of freedom (since the magnitude of weights can be any value between 0 and 1 or −1 and 0 before the sign(x) function is applied) and faster training.

Each penalty term is normalized for the size of the weight matrix ($m \times n$). For convolutional layers, the penalty function is applied to the semi-unrolled weight matrix $\mathbf{W}'$. Please note that the proposed loss function changes the spatial distribution of the weights, as opposed to the widely used L_1 and L_2 regularization, which are typically used to reduce overfitting by changing the distribution weights value.

The penalty function can now be combined with the original training objective, i.e., the loss function, to form the new training objective. It can be described as follows:

$$\text{Loss}'(\boldsymbol{\theta}, \mathcal{D}) = \text{Loss}(\boldsymbol{\theta}, \mathcal{D}) + \frac{\lambda}{L} \sum_{l=1}^{L} \text{Penalty}(\mathbf{W}_l). \tag{12.9}$$

where the scalar value $\lambda \in \mathcal{R}^+$ is the penalty rate and L is the number of penalized layers. The contribution of the penalty to the overall loss can be tuned by increasing or decreasing λ.

The definition of the loss function can be dynamically adjusted during training to influence the training target. For example, the loss function can be augmented by the penalty (as described in Algorithm 8) when the network starts to store weights of undesired values in the non-scrub and the scrub area else the weight update can be frozen. Such modification can reduce training time and is especially beneficial in CNN training, as the convolution operation is computationally more expensive compared to matrix-vector multiplication employed in MLP.

Algorithm 8 Algorithm for dynamically adjusting the loss function during training. (Non-) *Scrub Area Coverage SAC* (*nSAC*) is given as a percentage of the desired weights in SA (nSA). The parameter th is the corresponding augmentation threshold in percent, and η represents the learning rate of the gradient descent algorithm. For brevity, $\hat{\boldsymbol{\theta}}$ represents the other parameters besides the weights, e.g., $\mathbf{b}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$.

for epoch = 1 to epochss **do**
  if $SAC < T$ or $nSAC < T$ **then**
    $\text{Loss}'(\boldsymbol{\theta}, \mathcal{D}) \leftarrow \text{Loss}(\boldsymbol{\theta}, \mathcal{D}) + \frac{\lambda}{L} \sum_{l=1}^{L} \text{Penalty}(\mathbf{W}_l)$
    $\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial \text{Loss}'}{\partial \mathbf{w}}$ // update weights
    $\hat{\boldsymbol{\theta}}' \leftarrow \hat{\boldsymbol{\theta}} - \eta \frac{\partial \text{Loss}'}{\partial \hat{\boldsymbol{\theta}}}$ // update other parameters
  else
    $\text{Loss}'(\boldsymbol{\theta}, \mathcal{D}) \leftarrow \text{Loss}(\boldsymbol{\theta}, \mathcal{D})$
    $\mathbf{w}' \leftarrow \mathbf{w}$ // frozen weights
    $\hat{\boldsymbol{\theta}}' \leftarrow \hat{\boldsymbol{\theta}} - \eta \frac{\partial \text{Loss}'}{\partial \hat{\boldsymbol{\theta}}}$ // other parameters update
  end if
end for

Alternatively, the hyperparameter λ can be chosen before training to get desired *SAC* and *nSAC* coverage. Choosing a too high value of λ can overwhelm the loss and can lead to a lower initial t_0 accuracy. Similarly, smaller λ will lead to lower *SAC* and *nSAC* but higher t_0 accuracy.

In extreme cases, for example, $\lambda \to \infty$ or $th = 100$, scrub and non-scrub have only +1 or −1 weights. Therefore, in those cases, scrubbing is exact as it fully restores the intended weight matrices in the memristor crossbar array.

Self Learnable Scrub Area In the pre-defined scrub area method, we decide on the shape of the scrub area based on respective hyperparameters. Although, this method has negligible memory overhead, it gives NNs less freedom to choose the best scrub area for the task. In addition, it adds additional hyperparameters that need optimization. This can be problematic and too restrictive for larger networks on harder tasks. We, therefore, propose to let the NN define the position of the scrub area during training for CNNs.

To implement our approach, we re-parameterize the weight matrix $\mathbf{W}$ into $\tilde{\mathbf{W}}$, which is a constant ones matrix of shape $[C_{in} \times K_h \times K_w \times C_{out}]$, and smaller matrix Γ. Here, the Γ multiplies the $\tilde{\mathbf{W}}$ before each forward pass to obtain proxy weights, based on which the computation of a layer is calculated. The shape of the Γ matrix can be constructed in various ways, depending on the acceptable memory overhead and NN tasks. For example, a shape of $[C_{in} \times 1 \times C_{out}]$ will have learnable filters of 1s and -1s and memory overhead of $C_{out} \times C_{in}$. Therefore, the computation of a convolutional layer can be reformulated as:

$$\text{ConvolutionLayer}(\mathbf{x}) = (\tilde{\mathbf{W}}\Gamma)\mathbf{x} + \mathbf{b}. \tag{12.10}$$

In an extreme case, Γ will have a shape of $[C_{in} \times K_h \times K_w \times C_{out}]$, the same as the original weight matrix, leading to high scrub overhead. Therefore, we do not recommend such a shape.

For efficient learning, Γ needs to be initialized properly. We propose to initialize it as:

$$\Gamma = mean(|\mathbf{W}|). \tag{12.11}$$

Here, the mean for the initialization should be calculated in the same dimension as Γ, e.g., for Γ with a shape $[C_{in} \times 1 \times C_{out}]$, the mean should be calculated for each filter. Also, $\mathbf{W}$ is the original initialized (real-valued) weight matrix (before the proposed re-parameterization) with the state-of-the-art initialization method, e.g., Kaiming [249], initialization method. During training, Γ is binarized with the state-through-estimation technique (STE) [250] using the sign(x) function during the forward pass. STE is heavily used in the binary neural network to binarize both the weights and the activation values [44].

12.2.2.5. Hybrid Scrubbing: Boosting Scrubbing Accuracy At Initial Device Operational Time

If the scrub-prepared model (trained with either proposed method ① or ②) is mapped directly to the memristor-based crossbar arrays initially at time t_0, the inference accuracy can slightly reduce from the baseline accuracy at the same operational time, as shown in Section 12.2.3. In order to achieve high accuracy at both the initial and end of the expected device operational time, we proposed a hybrid scrubbing method that is used when the scrubbing is performed for the first time. In the naive approach, the complete NN needs to be stored in either the cloud or hardware and switched during the first scrubbing event, which is very costly. Alternatively, we proposed that the NN is trained with either the proposed method ① or ② then the layers not intended to scrub, e.g., first and last layers, are frozen and only the layers intended to scrub are freely trained. That way, the weight matrix of the layers need not be stored in either the cloud or hardware, and normal scrubbing can fully restore the scrub-prepared model. However, the mean μ and the variance σ of BatchNorm also have to be updated for proper normalization.

12.2.3. Results

12.2.4. Evaluation Setup

We have used MTJs as the memristor technology for detailed evaluations of our proposed scrubbing technique and uni-directional faults in the crossbar array. However, to reiterate, the proposed methods can be applied to mitigate retention faults in other memristor-based crossbar arrays. Compared to other

memristors, STT-MRAM has reached a comparably mature technology state, as demonstrated in industrial studies [251, 252].

Our training, fault-injection, and scrubbing simulation flow are shown in Figure 12.6. Faults are modeled as described in [29]. We have trained an MLP (six fully connected layers, 2048 neurons each) and a CNN (three Conv and three fully connected layers) with three different datasets: MNIST, Fashion-MNIST, and CIFAR-3. Our CNN topology is summarized in Table 12.6 and is based on the popular LeNet-5 topology. We have used an inflation ratio of three for MNIST, and five for Fashion-MNIST, which scales the output channels by that amount. In this section, we refer to the topology as LeNet-6. However, for the CIFAR-10 benchmark datasets, we use a nine-layer VGG based CNN with the same topology as [44], with the difference that the number of neurons is kept at 256 in the linear layers and an inflation ratio of one is used. We have used a Cosine Annealing learning scheduler [112], the ADAM optimizer with default settings [253], and the cross-entropy loss function.

Table 12.6.: Summary of MLP and CNN topology.

MLP	Neurons			2048
CNN	Convolution	Filters	LeNet-6	5, 2, 5
			VGG Block	2, 3
		Out Channel	LeNet-6	8, 16, 150
			VGG Block	128, 256, 512
		Stride		1
		Padding		1
	Fully-Connected	Neurons	LeNet-6	128
			VGG	256
	Avg-Pooling	Filters		2
		Stride		2

MNIST (handwritten digits) and Fashion-MNIST [254] have 28×28 gray-scale images representing handwritten digits ranging from 0 to 9 (10 classes), and 10 classes of clothes. Both datasets have 60K training images and 10K test images. We did not use any data-augmentation or pre-processing for these datasets. We use the CIFAR-10 dataset and additionally use also use a subset of the CIFAR-10 dataset with only three output classes (CIFAR-3). It has a 15K training set and a 3K test set, but CIFAR-10 has a 50k training and 10k test dataset. Each image of both the CIFAR-10 and CIFAR-3 consists of 32×32 RGB pixels. CIFAR-10 datasets are normalized and augmented with RandomHorizontalFlip with 50% probability and RandomCrop with a padding length of four on each border.

We have binarized our NN with the algorithm from [44], but initialized it with a pre-trained floating-point NN. The proposed cost function described in Section 12.2.2.4 is used during floating-point and BNN training. We have defined the shape of Γ as $[C_{in} \times 1 \times 1 \times C_{out}]$ because it provides a good balance between memory overhead and scalable scrub area.

The trained weight of the binary NN is mapped to different MTJ-based crossbars ($\mathcal{H}0$ to $\mathcal{H}8$ depending on the depth of the NN) for inference. The shape of all hidden layers is 2024×2024 for the MLP, and they are mapped to several MTJ-based crossbars with a dimension (m and n) of 256×256. The weight matrix splitting and current accumulation, as proposed in [64], is used in this section. In the evaluated CNNs, the shape of the weight matrices is different for different layers. Hence, they are mapped to either 64×64 or 128×128 crossbars. When the size of the weight matrix is bigger than the crossbar, they are split into several crossbars.

Due to hardware constraints, only a limited number of word-lines are activated concurrently for a larger crossbar, and the partial current sum is accumulated to get the total current sum of a layer. We activate either 32 or 64 word-lines concurrently, depending on the array size. The current sum of each post-synaptic neuron is scaled and shifted by pre-trained BatchNorm parameters. Those parameters are stored in the off-chip memory for reliability.

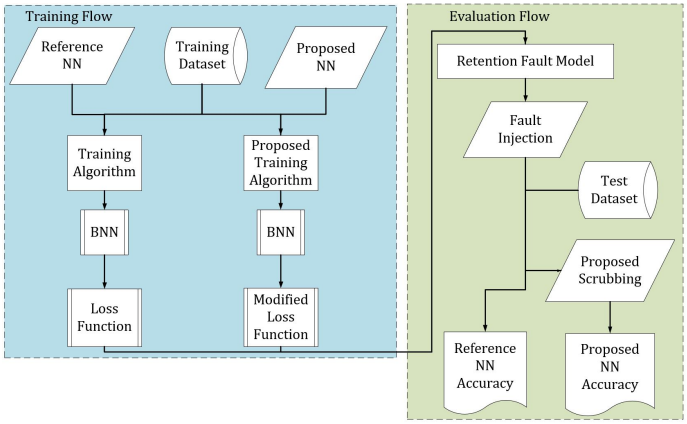


Figure 12.6.: Overview of NN training and evaluation flow.

The hidden layers have more parameters compared to the first and the last layer. The first layer's input and the last layer's activation are non-binary. As a result, this work is focused on retention faults in all Conv and hidden fully-connected layers only. For both MLP and CNN, we assume that the first and last fully connected layer's weights are mapped to a crossbar array ($\mathcal{H}0$ and $\mathcal{H}5$) with a high thermal stability factor $\Delta = 60$. Here, Δ represents the thermal stability factor.

Our system-level evaluation was performed using NVSim [160]. We parametrized the individually simulated STT-MRAMs for a 64×64 cell sub-array size, a 128×128 cell sub-array size, and a 2048×2048 cell subdivided into 256 × 256 matrix sub-arrays to reflect the shape of our evaluated neural network layers. The results are then used to evaluate the impact of different scrubbing rates. The MTJ device properties, operating temperature and other parameters for the evaluation are summarized in Table 12.7. For our evaluation, we have changed the thickness of the free layer to change the thermal stability of the MTJ device.

Table 12.7.: MTJ parameters and simulation setup

Parameter	Value
VDD	0.25V
Nominal Temperature	27°C
CMOS library	Globalfoundries 22FDX
MTJ radius	20 nm
RA	7.5 $\Omega\mu m^2$
TMR @ 0V	220%
'AP'/'P' resistance	19 kΩ/6 kΩ

12.2.4.1. Analysing Inference Accuracy with Scrubbing

We conducted two sets of experiments: baseline and proposed for each dataset. The retention faults are simulated in ten steps, each corresponding to 10% of the expected device operational time. The training algorithm of the baseline model is unaltered, and it is not scrubbed during the simulation. The proposed model is trained with Algorithm 8 for both CNN and MLP and is scrubbed frequently during the simulation so that faults do not accumulate over time. However, the effect of a faster or slower scrubbing frequency is analyzed in Section 12.2.4.5.

We experimentally show the benefits of the diagonal-shaped scrub area compared to the rectangular-shaped scrub area. In the case of a rectangular-shaped scrub area, the initial inference accuracy is significantly below (26.2%) the baseline accuracy, but the proposed diagonal-shaped scrub achieves an inference accuracy only below ~ 3% of the baseline accuracy as shown in Table 12.8. Furthermore, the rectangular-shaped scrub area has significantly more (4×) memory overhead, even for only two rectangles.

Table 12.8.: Comparison of the proposed diagonal and rectangular shaped scrub area for MNIST dataset with penalty function $\text{Penalty}_3(\mathbf{W})$. Evaluation is performed on a four-layer NN with 256 neurons per layer.

Scrub Shape	Inference accuracy (t_0)	Memory overhead\layer
None (Baseline)	97.38%	-
Diagonal	94.39%	1 point
Rectangular	71.1%	4 points

Table 12.9.: Evaluation of MNIST and Fashion-MNIST datasets with a smaller MLP model (four layers NN with 256 neurons layer). The thermal stability factor $\Delta = 30$ is chosen, and the evaluation is performed on both baseline and proposed model.

Dataset	Crossbar	accuracy at t_0		accuracy at t_{end}	
		Proposed	Baseline	Proposed	Baseline
MNIST	$\mathcal{H}1$	94.65%	97.35%	94.54%	7.52%
	$\mathcal{H}2$			94.71%	13.81%
Fashion-MNIST	$\mathcal{H}1$	85.11%	87.47%	85.01%	9.08%
	$\mathcal{H}2$			85.06%	11.31%

MLP For the MLP, a value of the threshold $th = 100$ and diagonal $d = 9$ are chosen before training to define the scrub area, and the results are summarized in Table 12.10. After training the proposed model, the scrub and the non-scrub area contain only the weights that are preferred by the proposed penalty function. Therefore, the initial inference accuracy (t_0) remains stable throughout the useful life of the device ($t_0 = t_{end}$). Our proposed scrubbing technique improves the inference accuracy by more than 82.89%, 74.46%, and 30.80% in MNIST, Fashion-MNIST, and CIFAR-3 datasets, respectively, over the expected device operational time.

When the number of neurons in the hidden layers is reduced to 256, and the number of layers is reduced to 4, the initial and the final inference accuracy of the proposed model still remains close to the baseline (fault-free) accuracy (2.81% in the worst case), as shown in the Table 12.9. Although the initial inference accuracy of the baseline model reduces, the proposed model still maintains similar inference accuracy compared to the 2048 neurons model (shown in Table 12.10). Therefore, our proposed approach is scalable to various MLP sizes and depths.

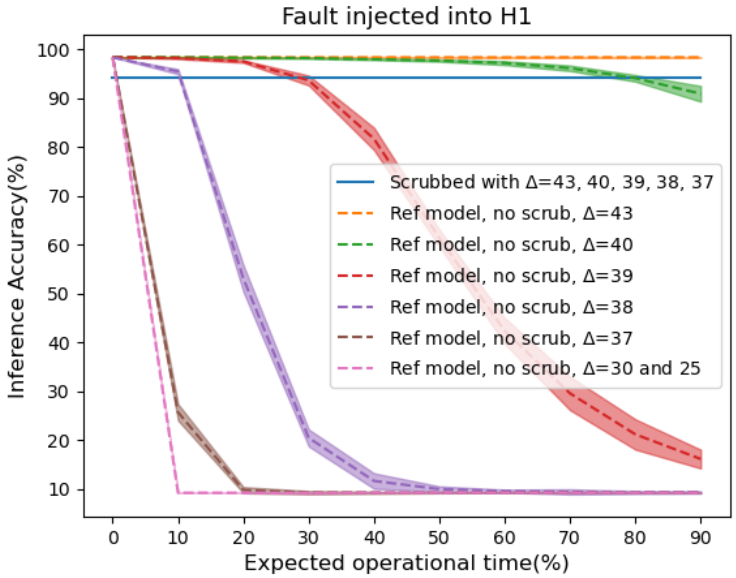


Figure 12.7.: Impact of different thermal stability factors on inference accuracy with MNIST dataset on MLP when the scrub prepared model is directly mapped to the initial time t_0. The uncertainty band shows $\pm 3\sigma_{acc}$, where σ_{acc} is the standard deviation of accuracy. **It is recommended to view this figure in color.**

Table 12.10.: Evaluation of different crossbars (one at a time) with our proposed scrubbing technique and the baseline model for MNIST, Fashion-MNIST, and CIFAR-3 datasets. $\Delta = 30$ is chosen, and the evaluation is performed on MLP.

Dataset	Crossbar	accuracy at t_0		accuracy at t_{end}	
		Proposed	Baseline	Proposed	Baseline
MNIST	$\mathcal{H}1$	94.08%	98.35%	94.08%	9.20%
	$\mathcal{H}2$				10.81%
	$\mathcal{H}2$				10.83%
	$\mathcal{H}2$				11.3%
Fashion-MNIST	$\mathcal{H}1$	84.46%	90.68%	84.46%	7.48%
	$\mathcal{H}2$				10.34%
	$\mathcal{H}2$				8.91%
	$\mathcal{H}2$				10.00%
CIFAR-3	$\mathcal{H}1$	64.2%	69.27%	64.2%	33.4%
	$\mathcal{H}2$				32.27%
	$\mathcal{H}2$				33.33%
	$\mathcal{H}2$				31.8%

Although the initial inference accuracy at t_0 is slightly better for the baseline model compared to our proposed technique, the inference accuracy is significantly better for our proposed scrubbing technique until the end of the expected device operational time (t_{end}). Please note that CIFAR-3 has only three output classes, and the probability that the output is correct at random is 33%, which is more than three times that of MNIST and Fashion-MNIST (10%). As a result, the inference accuracy degrades to around 33% from 69.27% under retention faults.

CNN For CNNs with the pre-defined scrub area (proposed method ①), a value of the threshold $th = 99$ is chosen before training, but raise $= d_1$ is selected depending on the shape and type of the layer. The hyperparameter d_1 for the Conv layer is calculated using $\frac{\text{rows}}{\text{cols}} = \frac{C_{in} \times K_h \times F_w}{C_{out}}$. The parameter run $= d_2$ is kept 1 to avoid repeated neurons. Furthermore, the diagonal $d = 1$ is chosen for the fully-connected layer. In the CIFAR-10 dataset, only the convolutional layers with more than 0.5 million parameters are scrubbed, as they have many parameters compared to other layers. Our proposed scrubbing technique achieves similar initial accuracy ($\sim 1-2\%$ below the training accuracy for the baseline model) but archives significantly higher inference accuracy at the end of the expected device lifetime, as summarized in Table 12.12. Our proposed method improves the inference accuracy by more than 87.76%, 78.02%, 60.46%, and 59.74% in the MNIST, Fashion-MNIST, and CIFAR-3 datasets, respectively, over the expected device operational time.

Table 12.11.: Approximate operational time of proposed Proposed and baseline model Baseline when the inference accuracy drops below the scrub prepared model. Here, $\mathcal{H}1$ crossbar is evaluated with $\Delta = 30$ on CIFAR-10 dataset.

Delta	Baseline	t^p
40	1.2 years	1.5 years
37	20 days	24 days
35	3 days	4 days
30	30 minutes	50 minutes

There are a few weights with undesired values in SA and nSA after training the proposed model. Undesired weights in SA and nSA refer to the weights that are not preferred by the proposed penalty function in the following. Hence, the scrubbing approximately restores the initial (t_0) inference accuracy.

For a harder task, e.g., the CIFAR-10, a more complex and freely expressible scrub area is required. We, therefore, used our self-learnable scrub area method ②. The results are summarized in Table 12.13. Similarly, our proposed method archives comparable accuracy initially at t_0 and significantly better accuracy compared to the baseline at the end of the expected device lifetime. Consequently, our method improves

accuracy by 70.97%. Since the learnable scrub area has a relatively higher scrub memory overhead, we recommend using this method for larger NN topologies and complex NN tasks. Otherwise, the scrub area can be predefined (proposed method ①).

Table 12.12.: Evaluation of different crossbars (one at a time) with our proposed scrubbing technique ① (pre-defined scrub area) in comparison to the baseline model. $\Delta = 30$ is chosen for the evaluation. Accuracy in the bracket shows the accuracy if the scrub prepared model is mapped at time t_0 (see Section 12.2.2.5).

Dataset	Crossbar	accuracy at t_0		accuracy at t_{end}	
		Proposed	Baseline	Proposed	Baseline
MNIST	$\mathcal{H}1$	98.97% (96.14%)	99.2%	96.0%	12.48%
	$\mathcal{H}2$			94.94%	10.43%
	$\mathcal{H}3$			95.91%	8.75%
Fashion-MNIST	$\mathcal{H}1$	89.72% (84.71%)	90.24%	84.47%	10.25%
	$\mathcal{H}2$			84.33%	6.31%
	$\mathcal{H}3$			84.58%	11.25%
CIFAR-3	$\mathcal{H}1$	91.73% (89.0%)	93.7%	89.03%	33.3%
	$\mathcal{H}2$			89.1%	35.2%
	$\mathcal{H}3$			89.03%	35.03%
	$\mathcal{H}4$			88.93%	28.47%

Table 12.13.: Evaluation of our proposed learnable scrub area method ② in comparison to the baseline model for the CIFAR-10 dataset on CNN topology. The evaluation is performed with a thermal stability factor $\Delta = 30$. Accuracy in the bracket represents the accuracy of the scrub-prepared model mapped at time t_0.

Dataset	Crossbar	accuracy at t_0		accuracy at t_{end}	
		Proposed	Baseline	Proposed	Baseline
CIFAR-10	$\mathcal{H}1$	83.3% (78.44%)	86.6%	78.44%	10.62%
	$\mathcal{H}2$			78.44%	7.93%
	$\mathcal{H}2$			78.44%	8.38%
	$\mathcal{H}2$			78.44%	7.47%

Impact of Variation in Thermal Stability Δ In a crossbar array, the Δ of MTJs can vary due to manufacturing variation, non-zero current during the read and write operation, and when the dimension of the device is scaled down. The Δ of the MTJs can be varied to evaluate the robustness of our proposed approach under different Δ.

The inference accuracy of the proposed method does not change with Δ as opposed to the baseline model given that the proposed method is scrubbed with a sufficiently high frequency, as shown in Fig. 12.7 for the MLP when the scrub prepared model is mapped to the crossbar, initially at t_0 time. In that case, when the thermal stability factor Δ is greater than 40, the baseline model performs slightly better, but our proposed scrubbing method performs significantly better in the long run when the thermal stability is below 40. We have observed the same behavior of the CNN when the thermal stability of the crossbar is varied.

On the contrary, if the hybrid scrub area model (as described in Section 12.2.2.5) is utilized initially t_0 and the NN is switched to the scrub prepared model at the first scrub event, a comparable inference accuracy can be achieved at all thermal stability (i.e., above or below the thermal stability value 40) by performing the first scrub event slower in comparison. In that case, when the first scrub event is performed, it is quite important to experience the benefit of high accuracy at the initial time t_0. The time when the first scrub event is performed should be chosen based on the retention capabilities of the memristor cells. For example, it takes around 1.2 to 1.5 years for the inference accuracy of the baseline and hybrid scrub area model to drop below the scrub prepared model, as shown in Table 12.11. However, at lower Δ, the inference accuracy can drop within 30 minutes. Therefore, a slightly higher accuracy of the baseline can be experienced for a shorter device operational time.

12.2.4.2. Analyzing the Effect of Frozen Layers During Training

In the case a threshold of $th = 100$ is chosen before training for the predefined scrub method, the scrub area and non-scrub contain only +1 and −1 weights, respectively. Consequently, the weights of the intended layers can be initialized with only +1 and −1 in the scrub and the non-scrub area, respectively, and then can be trained with Algorithm 8. NN only trains the biases b, weight scale Γ (optionally), and BatchNorm parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ of the frozen layers. This can significantly reduce the training time by reducing back-propagation calculations. Therefore, the training gets faster, and the hyperparameter search becomes less costly. We have found that comparable inference accuracy can be achieved with this method. In CNN, the accuracy at time t_0 of 96.9%, 83.85%, and 88.23% can be achieved for the MNIST, Fashion-MNIST, and CIFAR-3 datasets, respectively, with this method.

12.2.4.3. Analyzing Inference Accuracy with Different Scrub Areas

In the previous section, a fixed scrub area was evaluated for the pre-defined scrub area, but this can be changed before and after training by choosing a different value of d. If $d_1 \neq d_2$ for the Conv layers, then those values can be individually changed to define a different scrub area. A smaller scrub area is desirable in terms of scrubbing cost, as the content of fewer memristor cells must be restored.

Increasing or decreasing the diagonal d' from the value specified before training d will make the scrub area smaller or bigger. The smaller the scrub area ($d' > d$), the less synaptic +1 weights are corrected through scrubbing. On the other hand, with a larger scrub area ($d' < d$), more synaptic −1 weights are written over synaptic +1 weights due to scrubbing, as shown in Table 12.14 for both the MLP and the CNN with LeNet topology.

Due to the approximate nature of the NN, the inference accuracy only degrades noticeably after the scrub area is made significantly bigger ($d' \ll d$) or smaller ($d' \gg d$) compared to the original scrub area defined before training. This shows the robustness of the proposed approach. Due to parameter sharing, Conv layers have a significantly lower number of parameters and smaller crossbar shapes. Changing $d \rightarrow d'$ slightly after training causes a higher percentage of retention faults or −1 in the scrub area, which are scrubbed to +1 weights, as shown in Table 12.14. As a result, inference accuracy at t_{end} is degraded significantly.

Choosing a too high value for the NN training hyperparameter d before training for the pre-defined scrub area will lead to a weaker scrub area SA and massive t_0 inference accuracy degradation. The inference accuracy drops to 75.76%, and 93.10% for MNIST when a value $d = 1000$ and $d = 100$ were chosen, respectively, for the MLP with a 2048×2048 crossbar shape. The hyperparameter d should be chosen such that the size of the scrub and the non-scrub area are approximately equal to one another SA $\approx$ .nSA.

Table 12.14.: The effect of changing the diagonal scrubbing parameter on inference accuracy after training. Evaluated for MNIST dataset and $\mathcal{H}1$ crossbar with $\Delta = 30$.

	Diagonal	% of Retention Faults	% of -1 weights Scrubbed	Accuracy at t_{end}
MLP	100	8.73%	0%	90.01%
	50	3.98%	0%	93.73%
	9(original)	0%	0%	94.08%
	-50	0%	5.44%	91.78%
	-100	0%	9.51%	83.72%
CNN	4	28.27%	0%	11.66%
	3(original)	0%	0%	90.36%
	2	0%	33.33%	37.78%

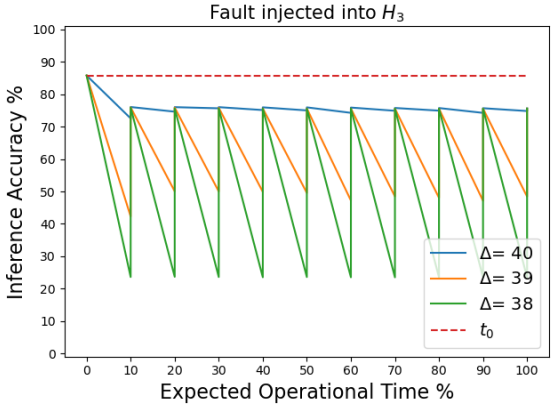


Figure 12.8.: The impact of scrubbing when scrub and non-scrub areas have undesired −1 and +1 weights, respectively. The crossbar $\mathcal{H}3$ is scrubbed with a scrubbing period f = 1y on the Fashion-MNIST dataset. **It is recommended to view this figure in color.**

12.2.4.4. Relaxing the Requirement of Scrub and Non-Scrub Areas

We analyzed the inference accuracy when the scrub and the non-scrub area only contain +1 and −1 synaptic weights, respectively, in Section 12.2.4.1 and 12.2.4.2. However, this requirement can be relaxed for the pre-defined scrub area without severely impacting the inference accuracy up to a certain point, as shown in Tables 12.15 and 12.16 for the MLP and the CNN, respectively.

When only the non-scrub area contains some undesired weights, the inference accuracy at the end of the expected device operational time degrades slightly for the MLP. We found that a higher number of undesired weights (0.80% ⟶ 2.65%) in the non-scrub area does not always lead to a lower final inference accuracy. In the case of CNN, allowing a smaller number of undesired weights in the scrub and the non-scrub area does not always lead to higher initial t_0 inference accuracy.

Allowing some undesired weights in both non-scrub and scrub areas does not guarantee a higher initial inference than allowing only undesired weights in the non-scrub area. In our case, we obtain slightly lower initial inference accuracy (87.16% ⟶ 85.81%) with MLP and similar initial inference accuracy with the CNN when evaluated on the Fashion-MNIST dataset and LeNet topology. In addition, the inference accuracy can degrade more at t_{end} compared to the case of only undesired weights in the non-scrub area, as shown in Table 12.15 for the MLP. This is due to the combined effect of not protecting synaptic +1 in the non-scrub and writing −1 weights in the scrub area to +1. However, in both cases, scrubbing will not restore the initial t_0 inference accuracy, as shown in Fig. 12.8 for different thermal stability factors Δ.

Table 12.15.: The result for the Fashion-MNIST dataset with thermal stability factor (Δ) = 30 when only non-scrub and both areas contain undesired weights. The evaluation is performed on MLP.

Crossbar	% of +1 weights in S'	% of −1 in S	Accuracy t_0	Accuracy t_{end}
Relaxed in Non-Scrub Area Only				
$\mathcal{H}1$	1.21%	0%	87.16%	86.37%
$\mathcal{H}2$	2.65%	0%	87.16%	87.03%
$\mathcal{H}2$	0.80%	0%	87.16%	86.91%
$\mathcal{H}2$	1.99%	0%	87.16%	87.13%
Relaxed in Scrub and Non-Scrub Area				
$\mathcal{H}1$	0.45%	1.54%	85.81%	80.61%
$\mathcal{H}2$	0.04%	2.81%	85.81%	85.67%
$\mathcal{H}2$	1.23%	3.16%	85.81%	75.58%
$\mathcal{H}2$	2.34%	4.52%	85.81%	75.95%

Table 12.16.: The result for the Fashion-MNIST dataset with thermal stability factor (Δ) = 30 when both scrub and non-scrub areas contain undesired weights. The evaluation is performed on CNN.

Crossbar	% of +1 weights in S'	% of -1 in S	Accuracy t_0	Accuracy t_{end}
Relaxed in Scrub and Non-Scrub Area				
$\mathcal{H}1$	0.93%	3.26%	77.48%	72.74%
$\mathcal{H}2$	2.94%	2.92%	77.48%	76.04%
$\mathcal{H}2$	0.63%	0.51%	77.48%	76.64%
$\mathcal{H}2$	0.10%	0.06%	77.48%	77.43%

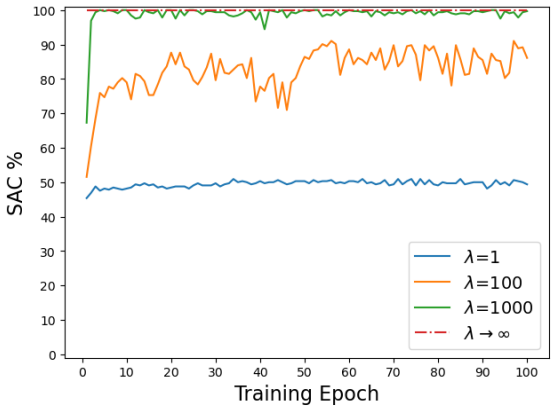


Figure 12.9.: The effect of penalty rate λ on *Scrub Area Coverage SAC* during training. The line with $\lambda \rightarrow \infty$ shows the case when the scrub and non-scrub areas are initialized with +1 and −1 weights, respectively. **It is recommended to view this figure in color.**

There is a trade-off between the initial t_0 inference accuracy and the final t_{end} inference accuracy. Although allowing some undesired weights in the scrub or non-scrub areas increases the initial t_0 inference accuracy, it can reduce the inference accuracy at the end of the expected device operational time (unless hybrid scrubbing is performed), especially when both scrub and non-scrub areas have undesired weights. For example, the inference accuracy of $\mathcal{H}2$ (75.58%) and $\mathcal{H}2$ (75.95%) at the end of the expected device operational time is lower in comparison (84.46%) to the result from Section 12.2.4.1 for the Fashion-MNIST dataset. The results suggest that the scrub and the non-scrub areas should not be relaxed at the same time and should not contain more than 3% of undesired weights, i.e., training the NN with Algorithm 8 and threshold $th = 97$ if the scrub prepared model is mapped at t_0 for the pre-defined scrub area.

Generally, initial inference accuracy can be increased by relaxing the requirement of the non-scrub area. We achieve a similar initial t_0 inference accuracy compared to the fixed scrubbing accuracy shown in Table 12.12 when the non-scrubbing area contains 25% to ∼ 40% undesired (+1) weights. This shows the trade-off between the number of layers scrubbed and inference accuracy.

As mentioned in Section 12.2.2.4, the requirement of the scrub and the non-scrub area can also be relaxed by choosing a small value of the penalty rate λ before training. Fig. 12.9 shows how the coverage of the scrub area changes during CNN training with different λ with the CIFAR-3 dataset. Table 12.17 shows how the trade-off between the initial t_0 and the final t_{end} inference accuracy was achieved with different λ values.

12.2.4.5. Analyzing Scrubbing Cost

To reiterate, the scrubbing is performed by re-writing the memristor cells in the scrub-area. Consequently, each scrubbing operation will consume some energy due to the write operation and have a certain delay. The cost of scrubbing in terms of total write energy and latency depends on the scrubbing period, the

Table 12.17.: The result for the CIFAR-3 dataset with thermal stability factor (Δ) = 30 when CNN trained with different penalty rate λ.

Lambda	SAC	Accuracy t_0	Accuracy t_{end}
1	50.0%	74.97%	33.6%
100	87.04%	64.13%	53.87%
1000	99.69%	63.9%	63.8%

number of memristor cells in the scrub region (crossbar size), memristor technology, and some design specification.

Choosing the right scrubbing frequency f is important to mitigate retention faults and reduce total write energy during scrubbing. Increasing the scrubbing frequency makes the crossbar more robust against retention faults, as shown in Fig. 12.11(a). When the scrubbing frequency is higher than the fault rate, a further increase in the scrub frequency f will not yield better fault tolerance but will only increase the total write energy, as shown in Fig. 12.10. Please note that we kept the write energy per cell constant to analyze the scrubbing cost in faster or slower scrubbing. However, when the thermal stability of the device is reduced, write energy also becomes lower.

Recall that, as mentioned in Section 2.10.2.3, retention time has an exponential relationship with temperature. Therefore, at high temperatures, the retention failure rate will increase. Hence, an MTJ-based crossbar with a low thermal stability factor or at a higher operation temperature will require more frequent scrubbing, as shown in Fig. 12.11(b) and (c). When the thermal stability of the crossbar decrease from 40 to 38, the inference accuracy decrease by about 5%, even though scrubbing is performed 2× faster.

The scrubbing latency depends only on the size of the scrub area, i.e., the number of cells to scrub. For our case, it takes 21.29μs to scrub one layer of the MLP with 2048 neurons. Here, the weight matrix of shape 2048×2048 is mapped into multiple smaller 256×256 crossbar arrays when $d = 9$.

Other memristor technologies, such as ReRAM have limited endurance and retention faults occur gradually depending on the rate of conductivity drift. Therefore, less frequent scrubbing needs to be performed.

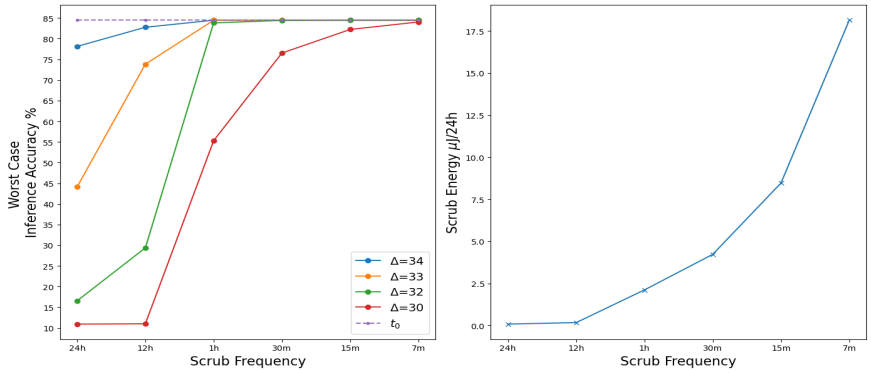


Figure 12.10.: a) Shows the relationship between scrub frequency and worst-case inference accuracy. When the scrub frequency is high, inference accuracy does not degrade from the initial t_0 inference accuracy (the worst case inference accuracy = initial t_0 inference accuracy), b) Shows the relationship between scrub frequency and total energy µJ/24h. Scrub energy increases with scrub frequency as more scrubbing is performed. **It is recommended to view this figure in color.**

12.2.5. Discussion

We have experimentally observed that the scrub area SA and the non-scrub area nSA of the crossbar have $\sim 40-60\%$ +1 and $\sim 40-60\%$ −1 weights, respectively. Hence, $40-60\%$ of +1 weights in the non-scrub area nSA are subject to faults, and the same amount of −1 weights in the scrub area SA are

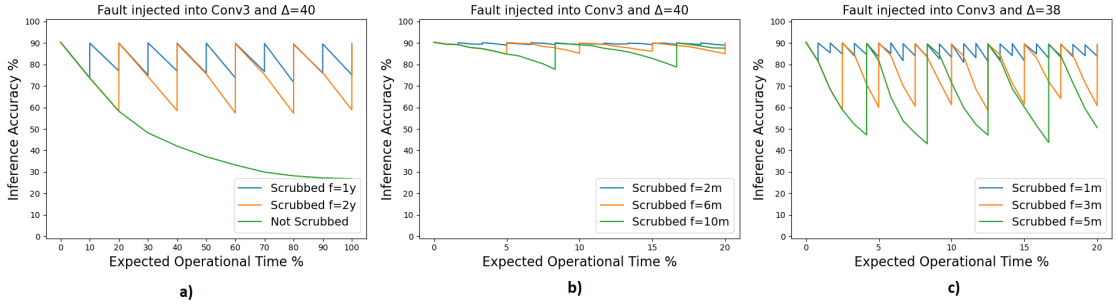


Figure 12.11.: a) Impact of two different scrubbing periods (f) and not scrubbing on inference accuracy with MNIST dataset with a thermal stability factor $\Delta = 40$. b) & c) shows the relationship between the thermal stability factor, scrubbing period, and inference accuracy. Fault injected into the third convolution (Conv3) layer of CNN. Evaluation is performed on the proposed model with the MNIST dataset. **It is recommended to view this figure in color.**

overwritten to +1 through scrubbing. The combined effect of this will lead to a massive inference accuracy degradation. Hence, another challenge is to ensure the scrub SA and non-scrub nSA area of the crossbar predominantly contain +1 and −1 weights, respectively. While some −1 weights that may have been stored in the scrubbing region SA will be overwritten to +1s, we hypothesize that having only a few of these faults will not significantly impair the inference accuracy. Therefore, we propose the use of a penalty function (see Section 12.2.2.4) during training to encourage the NN to respect the scrubbing and non-scrubbing regions, respectively.

In a common crossbar structure, interconnects are fixed, and the output of a layer L_i is the input of the next layer L_{i+1}. Hence, the rearrangement of weights post-training to map all +1 and −1 to the scrub SA and the non-scrub nSA region, respectively, is impractical for a common crossbar architecture, because the relationship between the output of a layer and the input of the following layer cannot be maintained for all the layers.

Our approach is generalizable to mitigate retention faults in any memristor-based crossbar. However, MTJ-based crossbars are used as the case study. In the case of other memristive technologies, the mapping of +1 and −1 weights in the crossbar can be adjusted depending on the direction of the uni-directional state change.

12.2.6. Scientific Impact of This Work and Contributions

The scientific impact of this work and the main contributions can be summarized as follows:

- **Aging and Thermal Stability Induced Uncertainty:** This work analyzed and showed the uncertainty in weights of an NN mapped to memristor-based CiM architectures. Specifically, the state of a memristor cell can suddenly change at a given time, leading to uncertainty in weight storage.
- **Novel Scrubbing Technique:** A novel approximate scrubbing method specifically designed to mitigate retention faults in memristor-based CiM architectures. A thorough evaluation of the proposed method showed a significant improvement in the reliability of CiM architectures by restoring the intended weight matrices with virtually no storage overhead.
- **Retention Aware Training:** A novel NN training technique is proposed that adjusts the weight matrix to our scrubbing requirement during training. Several optimization objectives are proposed depending on the scrubbing techniques and task difficulty.
- **Retention Aware Parameter Mapping:** A novel NN parameter mapping technique or weight organization is proposed. Specifically, all the stable and unstable memristor cells are proposed to map to predefined regions of the crossbar, respectively.

- **Scalability in Memristor Technologies:** The proposed approach is applicable to any memristor technology, i.e., potentially a memristor technology-agnostic. Technology-dependent scrubbing is discussed, including weight mapping to the crossbar array and mortifying the rate of scrubbing. However, retention faults in STT-MRAM occur suddenly, which is more relevant for this approach.
- **Algorithm-Hardware Co-Optimization:** Several algorithmic optimizations are done for minimal impact on computational, performance, and memory overhead in hardware, such as reduction scrubbing cost and latency. Also, hardware optimization, such as scrubbing controller design, is performed to meet the scrubbing need.
- Environmental Impact: The proposed approach improves longevity NN acceleration in CiM architectures efficiently. Specifically, memristive cells with an extremely short retention time can be reused instead of discarded. Therefore, our approach indirectly contributes to sustainability by encouraging less wasteful practices, i.e., in electronic device usage and management.
- **Broader Applicability in AI Accelerator:** Although the focus of this work is on CiM architectures, the proposed approach can be adapted for other types of AI systems where retention faults are a concern.

12.2.7. Section Summary

In this section, we proposed an approximate scrubbing technique to mitigate retention faults caused by aging or external environmental factors in resistive memristor-based NN accelerators. Our proposed technique divides the crossbar into a scrub and non-scrub area. Different shapes of the scrub area are proposed depending on the type of NN and the difficulty of the task. Specifically, unstable cells are mapped to the scrubbing region, and a scrubbing controller re-writes the scrubbing region periodically with an optimal scrubbing period to mitigate retention faults. We introduced an optimization technique during NN training to minimize the number of unstable and stable cells in the scrubbing and non-scrubbing regions, respectively. Our proposed scrubbing technique improves the inference accuracy over the expected device lifetime up to 85.51% for MLP and 87.77% for CNN with virtually zero memory overhead. It enables higher memory density of the crossbar by reducing the size of the memory cells and a reduction in write latency and energy of certain memristors without trading off retention time.

12.3. Guaranteed Soft-Faults Correction for Digital AI Accelerators

Error correction codes (ECCs) are widely used in conventional memories to guarantee fault-free operation up to a certain fault rate. However, this comes with considerable storage overhead as well as encoding and decoding overhead (area, power, and latency). Since NNs can have millions of parameters, the use of ECC can lead to substantial memory overhead, which can make it impractical for many resource-constrained edge AI accelerators.

In this section, we propose *the NN-ECC* method, a generalized way to eliminate the ECC parity storage overhead (*zero memory overhead*) of different linear block ECC schemes for the memories storing NN model parameters. The proposed method takes advantage of the fact that the memories in NN accelerators are used to store the model parameters. This provides the opportunity to embed the ECC parity bits alongside the data bits without extending the memory size. Therefore, we proposed to embed ECC parity bits directly into the NN weight during training using a *multi-task learning algorithm* without increasing the size of the original weight matrices while achieving baseline accuracy. As a result, both the data and the parity bits are integral parts of the weight matrices and actively contribute to learning and inference tasks. In our approach, ECC encoding is done off-chip and during training. Thus, no encoding operation is required during inference, and the decoding process is the same as ECC for conventional memories. This paper specifically addresses the correction of weights stored in the memory of digital hardware-based NN accelerators (GPUs, TPUs, FPGAs, and digital ASICs) to prevent performance degradation.

The overall contribution of our approach is summarized as follows: 1. incorporating ECC parity bits into the weight matrix without increasing its size so that the size of memories storing NN model parameters remains the same, 2. a multi-task learning algorithm that eliminates the need for specific weight distribution and higher quantization levels for embedding ECC parity into the weight matrix, 3. provision for both the single and multi-bit error correction with zero memory overhead for ECC parity bits, 4. elimination of extra memory overhead required to identify the parity location during decoding, and 5. capability to perform error detection and correction not only before the inference but also during the inference.

This section is based on our publications IEEE VTS24 [255]. However, the contributions related to the proposed ECC design for zero overhead are not a contribution of this thesis.

12.3.1. Problem Definition

In this section, *we directly embed the ECC parity bits into the NN weight matrix instead of using extra memory bits to store them.* Unlike traditional ECC for memory applications, encoding of weight bits is *not* done in hardware, but in software during training. The decoding follows traditional ECC logic, but with a smaller code size $((n^{\bar{p}} = k, k^{\bar{p}}))$ than traditional code size $((n, k))$. Consider a weight matrix $W_{d_1 \times d_2}$ where d_2 represents the size of the data word or the number of data columns in the memory array that stores the weight parameters. A key constraint is that the size of the weight matrix, i.e., the size of memory, remains the same after ECC encoding.

12.3.2. methodology

12.3.2.1. Encoding of Weight matrix using ECC

In a typical ECC design for fault-tolerant NN applications, the weight bits of the NN serve as a data word as done in [141], and the computation of parity information involves performing modulo-2 multiplication between the weight bits and the generator matrix of the ECC scheme. The output of the weighted sum operation of NN is taken column-wise, so the redundant-parity bits can be added as extra columns in a weight matrix. The number of columns in the weight matrix decides the size of the data word, and

accordingly, the coding dimension can be chosen for error detection and correction. Let us have a weight matrix W of linear layer l with shape $d_1 \times d_2$. The encoded weight matrix, $[\boldsymbol{W_c}]_{d_1 \times n} = [W|W_{\bar{p}}]$ with the code (n, k), is defined as:

$$\boldsymbol{W_{\bar{c}}} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1d_2} & w_{\bar{p}_{11}} & \dots & w_{\bar{p}_{1,n-k}} \\ w_{21} & w_{22} & \dots & w_{2d_2} & w_{\bar{p}_{21}} & \dots & w_{\bar{p}_{2,n-k}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ w_{d_1 1} & w_{d_1 2} & \dots & w_{d_1 d_2} & w_{\bar{p}_{d_1 1}} & \dots & w_{\bar{p}_{d_1,n-k}} \end{bmatrix} \tag{12.12}$$

Where, $[W_{\bar{p}}]_{d_1 \times (n-k)}$ contains the parity information, $k = d_2$ is the length of the data word, $[W_{\bar{c}}]_{d_1 \times n}$ is the encoded weight matrix with $n >> d_2$.

We designed a new ECC with dimensions $(n^{\bar{p}}, k^{\bar{p}})$ to satisfy the given requirement of maintaining the size of memory even after ECC encoding. The generator matrix $(\bar{G})$ of the newly obtained ECC $(n^{\bar{p}}, k^{\bar{p}})$ is used to encode the weight matrix $[W]_{d_1 \times d_2}$ to give an encoded weight matrix $[W'_{\bar{c}}]_{d_1 \times n^{\bar{p}}}$ whose shape is equal to the original weight matrix W. The newly encoded weight matrix, $[W'_{\bar{c}}]_{d_1 \times n^{\bar{p}}} = [W'_k|W'_{\bar{p}}]_{d_1 \times n^{\bar{p}}}$, contains both data and parity information, where $n^{\bar{p}} = d_2$.

$$\boldsymbol{W'_{\bar{c}}} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1k^{\bar{p}}} & w_{\bar{p}_{11}} & \dots & w_{\bar{p}_{1,n^{\bar{p}}-k^{\bar{p}}}} \\ w_{21} & w_{22} & \dots & w_{2k^{\bar{p}}} & w_{\bar{p}_{21}} & \dots & w_{\bar{p}_{2,n^{\bar{p}}-k^{\bar{p}}}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ w_{d_1 1} & w_{d_1 2} & \dots & w_{d_1 k^{\bar{p}}} & w_{\bar{p}_{d_1 1}} & \dots & w_{\bar{p}_{d_1,n^{\bar{p}}-k^{\bar{p}}}} \end{bmatrix} \tag{12.13}$$

To ensure that the ECC covers the entire weight matrix, the shape of $W'_{\bar{c}}$ is kept equal to the original weight matrix W. The computation of $W'_{\bar{c}}$ is done by first selecting a $k^{\bar{p}}$ number of columns from W, defined as W'_k. Then the parity matrix $W'_{\bar{p}}$ is computed by modulo-2 matrix multiplication of W'_k and the generator matrix $(\bar{G})$ of the ECC code $(n^{\bar{p}}, k^{\bar{p}})$.

In this section, we apply ECC per 64 weight bits, i.e., equivalent to 64 columns, as demonstrated in previous research work [140, 141]. However, our method also applies to other data sizes, such as 128 bits, 256 bits, etc. We employ two widely used linear block ECC schemes: Hamming and BCH codes. The new code dimension can be computed according to the discussion in Section 2.10.6. Figure 12.12 illustrates the proposed NN-ECC encoding process using the Hamming code to protect 64-bit weights. In the case of conventional ECC encoding, (n=72, k=64) code is used to protect 64-bit weight requiring additional 8 parity bits. However, our proposed approach uses a more compact code size of (n=64, k=57). This ensures that the encoded weight bit size remains unchanged from the original, eliminating the memory overhead for ECC parity bits.

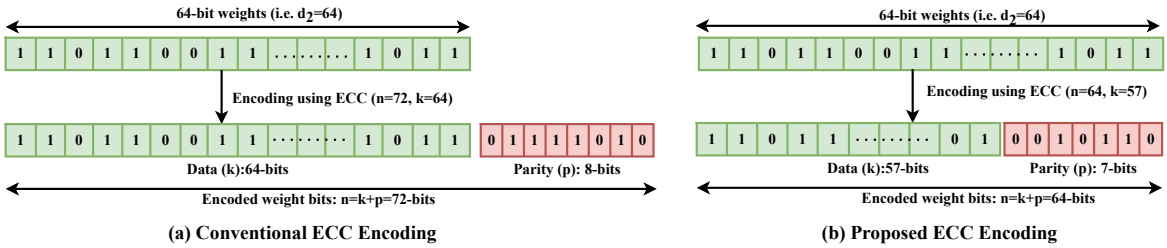


Figure 12.12.: Example of proposed NN-ECC encoding process showing the size of original weights and encoded weights are equal.

12.3.2.2. Decoding Process for Error Correction

The decoding process of the proposed NN-ECC follows an error detection and correction approach similar to conventional ECC methods used for memory applications [90, 91], and is consistent with existing strategies used for ECC parity overhead reduction [140, 141]. However, the proposed NN-ECC uses smaller code size, such as $(n^{\bar{p}} = 64, k^{\bar{p}} = 57)$ rather than $(n = 72, k = 64)$ SEC-DED Hamming code, as illustrated in Fig. 12.12. This reduced code size might result in slightly reduced decoding complexity. The decoding process involves modulo-2 matrix multiplication between the encoded weight bits and the ECC parity check matrix H to calculate the syndrome vector. Subsequently, error localization and correction can be carried out based on the value of the syndrome vectors. The decoding overhead trends would be similar to the conventional ECC used for the memory applications [90, 91, 256, 257, 258].

The existing approach [140, 141] performs the error correction before inference, as they need to mask the parity bits before inference starts to restore the original weight bits occupied by the parity bits. In contrast to the existing method [140, 141], there is no need for parity masking before inference starts in our approach because both the data bits and the parity bits are inherent components of the weight matrix and actively contribute to the inference process. Due to this, the proposed method offers the advantage of concurrently performing error detection and correction during the inference process also.

12.3.2.3. In-training ECC embedding using multi-task learning

NN learns its parameter θ for a functional task defined by the dataset $\mathcal{D}$ by minimizing a loss function $\mathcal{L}$. However, for the proposed NN-ECC, the NN should learn not only functional tasks but also fault-tolerant tasks, such as incorporating the ECC parity information directly into the weight matrix without increasing the size. The fault-tolerant task can be defined as optimizing a subset of the θ that fulfills the requirement of ECC encoding without needing any additional data. The overall parameter for both tasks is defined as $\bar{\theta}$. In the proposed multi-task learning, we ensure that the NN learns the function task and the fault-tolerant task without performance degradation. To achieve this, we update $\bar{\theta}$, which involves both the ECC parity and the data parts during NN training. As a result, our method jointly learns the shared parameters $\bar{\theta}$ using the same data.

To obtain the encoded weight matrix, the generator matrix must be multiplied (using the modulo-2 operation) with each row of the weight matrix. However, the weight matrix of the convolutional layers is 4D in shape $[C_{out} \times C_{out} \times K_H \times K_W]$. Here, C_{in}, C_{out}, F_H, and F_W refer to the input channels, the output channels, the height, and the width of the kernels. To satisfy the matrix-vector multiplication (MVM) rule, the four-dimensional weight matrix is reshaped into a two-dimensional weight matrix of shape $[C_{out} \times (C_{in} \cdot F_H \cdot F_W)]$. Here, $\cdot$ denotes arithmetic multiplication. However, the computational cost of such an MVM can be expensive as the number of rows is significantly high. To solve this problem, we have reshaped the weight matrix again by inserting an additional dimension of size one into the row position to give a new shape $[C_{out} \times 1 \times (C_{in} \cdot F_H \cdot F_W)]$. This replaces MVM with matrix-matrix multiplication between the generator and the weight matrix. Note that, this does not effect computation during inference and MVM represent a vectorized version of MAC operation.

We have used common loss functions, e.g., cross-entropy for classification, to learn both tasks. At the initialization of the weight, parity is embedded in the weight matrix, and it can be described as $\hat{\boldsymbol{W}} = f_{ECC}(\mathbf{W}, \bar{G})$. Here, the function $f_{ECC}(.)$ describes the encoding of the weight matrix as described in Section 12.3.2.1. The weighted sum is performed on the encoded weight matrix during the forward pass. Since the loss is calculated with respect to the encoded weight matrix, NN optimizes both tasks.

During backpropagation, the NN parameters are updated with the gradient descent algorithm for the functional task. Subsequently, the NN parameter is updated deterministically using $f_{ECC}(.)$. Since the ECC embedding preserves the data part of the weight matrix and the gradient is not used in weight update, their gradient should not affect the backpropagation for the functional task. As a result, the straight-through estimator (STE) [250] is used during forward and backward propagation.

With STE, during weight update, ECC embedded weights are computed by applying the $f_{ECC}(.)$ function to their corresponding functional weight proxies. During backpropagation, a gradient is an estimation of functional weight proxies, i.e., the STE simply treats the gradient of the function $f_{ECC}(.)$ with respect to functional weight proxies as an identity and can be defined as $\frac{\delta W}{\delta W'}$=1. Therefore, STE allows us to estimate the gradient of the loss with respect to the function $f_{ECC}(.)$ and update the proxies:

$$\frac{\delta \mathcal{L}}{\delta W'} = \frac{\delta \mathcal{L}}{\delta W} \tag{12.14}$$

The overall algorithm for training is described in Algo. 9.

Algorithm 9 Proposed Multi-task learning algorithm

Require: initial θ, loss function $\mathcal{L}$, total number of layers L, NN model $\mathcal{F}$, Code dimension $(n^{\bar{p}}, k^{\bar{p}})$, Generator matrix $\bar{G}$, training dataset $\mathcal{D}_{train} \subset (X, Y)$.
Result: trained parameters $\bar{\theta}$ for $(n^{\bar{p}}, k^{\bar{p}})$ ECC code
Initialization of NN parameters
Initialize parity subset of NN parameters using $f_{ECC}(.)$
for epoch **in** epochs **do**
  Sample random minibatch $\mathcal{D}_d$
  Forward pass $\mathcal{D}_d$ on NN with the ECC embedded parameter $\bar{\theta}$
  Calculate the loss $\mathcal{L}$ on the embedded ECC parameter $\bar{\theta}$
  Backpropagate and update parameters $\bar{\theta}$
  Flatten the weight matrix of the convolutional layers into a matrix of shape $[C_{out} \times (C_{in} \cdot F_H \cdot F_W)]$
  Reshape flattened weight matrix into shape $[C_{out} \times 1 \times (C_{in} \cdot F_H \cdot F_W)]$
  Update the reshaped weight matrix $\boldsymbol{W}'_l, \forall l \in [0, \cdots L]$ using $f_{ECC}(.)$
  Reshape update weight matrix to its original shape $[C_{out} \times C_{in} \times F_H \times F_W]$
end for

12.3.3. Evaluation

12.3.3.1. Simulation Setup

To enable the fault-tolerance in NN using by mean of error correction using ECCs, we explore the effectiveness of two widely used linear block ECCs: Hamming code and BCH code, both having t error correction capability and $t + 1$ error detection capability. We consider the BCH code with various error correction capabilities, including $t = 2$, $t = 3$, and $t = 4$, to highlight the versatility and ability of the proposed NN-ECC to accommodate a higher number of parity bits to enable multi-bit error correction capability. The proposed NN-ECC is evaluated using state-of-the-art NN models with 8-bit quantized weights on the CIFAR-10 and CIFAR-100 datasets. These models, along with their number of parameters, are detailed in Table 12.18.

Table 12.18.: NN Models and Their Respective Parameter Counts.

NN Model	ResNet-18	EfficientNet-B0	MobileNet-V2	ResNet-34
#Parameters	89.3×10^6	43.7×10^6	17.1×10^6	17.9×10^6

We train each model once without embedding any ECC during training and four times to embed different ECC schemes with different error correction capabilities ($t = 1$ to 4) using the proposed Algorithm 9. We have used ADAM optimization with the default setting in PyTorch and a cross-entropy loss function. The model that achieved the best accuracy in the validation dataset is used for the simulation.

In the context of fault injection, bit-flip faults are introduced into the weights of NN models by randomly sampling from a uniform distribution before inference. Following the fault injection phase, error correction is executed based on ECC schemes. The fault tolerance is evaluated by observing the improvement in the NN inference accuracy with the proposed NN-ECC solution. We have performed 100 Monte Carlo simulations and reported the mean accuracy.

12.3.3.2. ECC dimension ($n^{\bar{p}}, k^{\bar{p}}$)

The possible code dimensions $(n^{\bar{p}}, k^{\bar{p}})$ for the Hamming code ($t = 1$) and the BCH code ($t = 2, 3, 4$) with different column shapes (d_2) of the weight matrix are presented in Table 12.19. In this paper, we have used the $(n^{\bar{p}} = 64, k^{\bar{p}} = 57)$ code for the Hamming code and $(n^{\bar{p}} = 64, k^{\bar{p}} = 51)$, $(n^{\bar{p}} = 64, k^{\bar{p}} = 45)$, and $(n^{\bar{p}} = 64, k^{\bar{p}} = 39)$ for the BCH code with $t = 2, 3, 4$ error correction capability, respectively, to ensure error correction per 64-bit weights. In this way, the proposed approach ensures that the dimension of the encoded weight matrix perfectly matches that of the original weight matrix, resulting in a complete elimination of the additional memory cells required to store the ECC parity bits, representing a 100% memory overhead reduction.

Table 12.19.: The possible Code Dimension $(n^{\bar{p}}, k^{\bar{p}})$ for Different Column size (d_2) of the Weight Matrix for Proposed NN-ECC

ECC	Hamming (t=1)	BCH (t=2)	BCH (t=3)	BCH (t=4)
d_2	$(n^{\bar{p}}, k^{\bar{p}})$	$(n^{\bar{p}}, k^{\bar{p}})$	$(n^{\bar{p}}, k^{\bar{p}})$	$(n^{\bar{p}}, k^{\bar{p}})$
64	(64, 57)	(64, 51)	(64, 45)	(64,39)
128	(128, 120)	(128, 113)	(128, 106)	(128, 99)
256	(256, 247)	(256, 239)	(256, 231)	(256, 223)

12.3.3.3. Impact on Baseline Accuracy due to ECC Encoding

It is essential to assess how our proposed ECC encoding techniques based on multi-task learning impact NN baseline accuracy. Table 12.20 and 12.21 show the impact of the proposed ECC encoding techniques based on multi-task learning on NN baseline accuracy. Our method achieves accuracy comparable to the baseline across diverse NN topologies and various ECC configurations in CIFAR-10 and CIFAR-100 datasets. Furthermore, the encoding using the multi-bit ECC such as BCH code with $t = 4$ also shows a negligible drop in accuracy, less than 1% for the CIFAR-10 dataset and up to $\sim$ 2% for the CIFAR-100 dataset. This is achieved despite the fact that BCH code requires a large number of parity bits, posing challenges in simultaneously meeting functional tasks and embedding ECC parity bits into the weight matrix without increasing the size of the weight matrix.

Table 12.20.: Impact on Baseline accuracy due to ECC Encoding (CIFAR-10).

Topology	NN Inference Accuracy (%)				
	Baseline	Proposed NN-ECC			
		Hamming (t=1)	BCH (t=2)	BCH (t=3)	BCH (t=4)
ResNet-18	92.05%	92.03%	92.27%	92.44%	92.50%
EfficientNet-B0	90.19%	89.95%	90.30%	89.44%	89.20%

12.3.3.4. Fault-tolerant Analysis with Proposed NN-ECC

Fig. 12.13 and Fig. 12.14 show the result of random fault simulation for different NN models with CIFAR-10 and CIFAR-100 datasets. The baseline accuracy drops very sharply, even at a very low fault rate. However, the accuracy remains stable up to a certain fault rate when the error is corrected using Hamming and

Table 12.21.: Impact on Baseline accuracy due to ECC Encoding (CIFAR-100).

Topology	NN Inference Accuracy (Top-1 %, Top-5%)				
	Baseline	Proposed NN-ECC			
		Hamming (t=1)	BCH (t=2)	BCH (t=3)	BCH (t=4)
ResNet-34	(72.5, 91.5)	(72.3, 91.4)	(72.3, 91.5)	(71.6, 90.9)	(71.6, 90.4)
MobileNet-V2	(67.7, 89.3)	(68.0, 88.3)	(66.2, 89.1)	(67.3, 88.8)	(65.6, 88.7)

BCH codes. As the error correction capability increases, the sustainability of NN against the number of injected faults also increases significantly. The increment in the fault tolerance limit almost doubles with the increase in the error correction capability by 1 bit. For example, the fault tolerance limit of the BCH code with t=2, t=3, and t=4 is almost 2×, 4×, and 8× the Hamming code ($t = 1$), respectively, as shown in Fig. 12.13 and Fig. 12.14. In EfficientNet-B0, the improvement is minor, as there was not much reduction in baseline accuracy, as shown in Fig. 12.14. However, if fault injections target sensitive weights (beyond the scope of this work), the proposed strategy could play a crucial role in recovering sensitive weights to avoid any drastic accuracy drop.

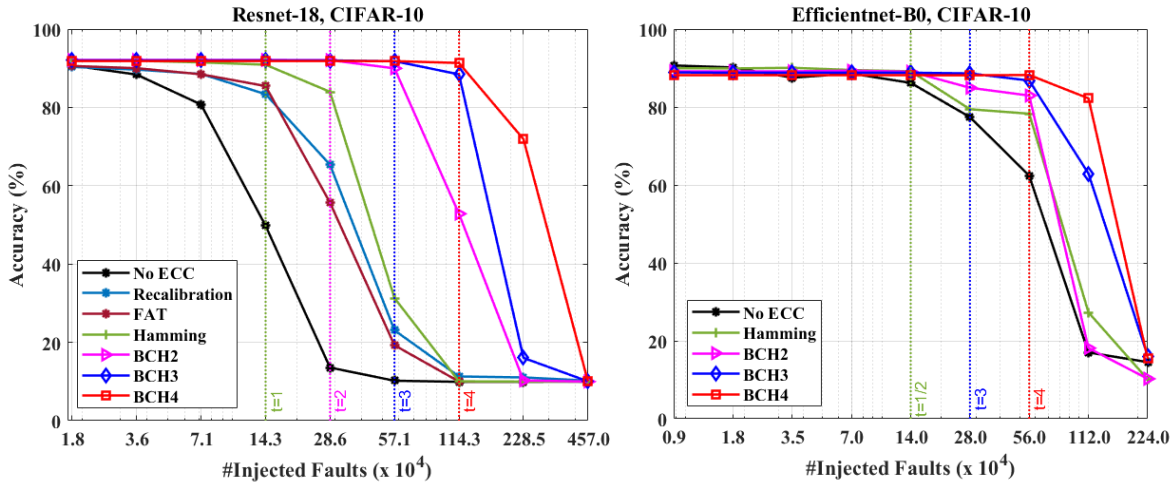


Figure 12.13.: NN inference accuracy on CIFAR-10 Datasets. The vertical dotted line indicates the fault-tolerance limits with t number of error corrections. **It is recommended to view this figure in color.**

Furthermore, we also demonstrate a case of the superiority of the proposed NN-ECC against fault-aware training (FAT) and recalibration-based fault-tolerant strategy [128], as shown in Figure 12.13 (ResNet-18 on CIFAR-10). We consider a case of FAT and recalibration for a specific fault rate (7.1×10^4 number of faults), and we observe a noticeable reduction in starting accuracy. Subsequently, the accuracy experiences a significant decline after reaching the designated fault rate, as shown in Figure 12.13. These strategies recover the accuracy *approximately* up to the designated fault rate, and they are computationally intensive, which becomes even more pronounced with higher fault rates. Furthermore, they may lead to a greater loss in starting accuracy at higher fault rates.

12.3.3.5. Comparative Analysis with ECC-based Related Work

The proposed NN-ECC offers notable benefits over existing state-of-the-art ECC parity overhead reduction techniques [138, 139, 140, 141], as highlighted in Table 12.22. A detailed discussion in this context is followed next.

Our approach does not require a specific symmetric weight distribution and a higher quantization level to store the parity. Instead, we employ multi-task learning to embed the parity into the weight matrix

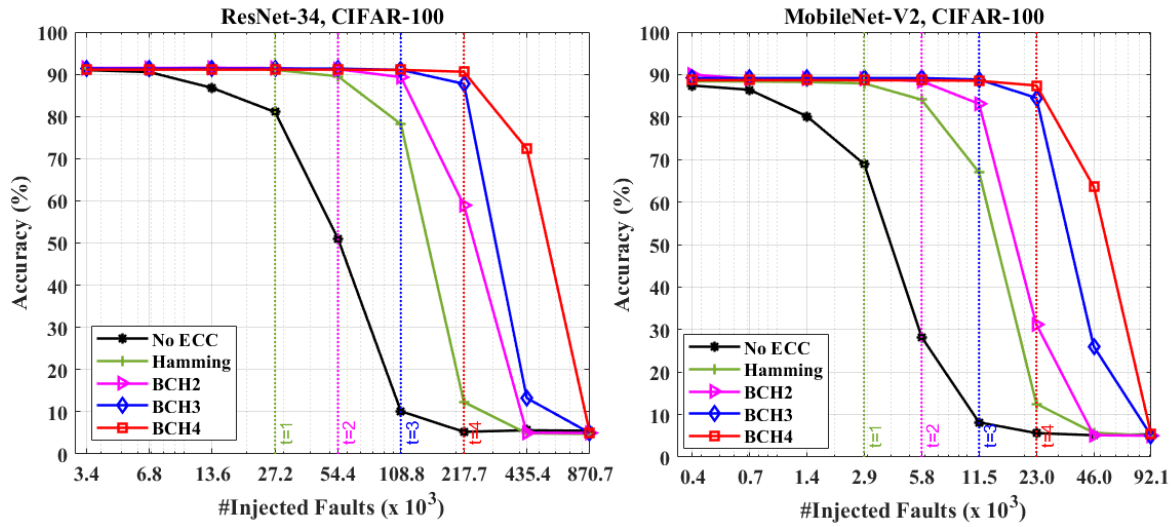


Figure 12.14.: NN inference accuracy (Top-5) on CIFAR-100 Datasets. The vertical dotted line indicates the fault-tolerance limits with t number of error corrections. The fault-tolerance trend was similar in the case of Top-1 accuracy. **It is recommended to view this figure in color.**

without expanding its dimensions. Also, the proposed ECC encoding, based on multi-task learning, does not degrade the baseline inference accuracy ($\sim$ 2%), even for BCH codes with $t = 4$ error correction capability, which requires a very high number of parity bits. On the contrary, the work in [141] can incur a drop in accuracy of up to $\sim$ 4% due to the ECC encoding even for $t = 2$ error correction, which might worsen with asymmetry in weight distributions.

Secondly, the proposed NN-ECC possesses the capability to perform not only single-bit but also multi-bit error correction, even more than two with zero memory overhead, as illustrated in Table 12.23, which provides a comparative analysis of memory overhead for various ECC scheme. This level of flexibility and error correction capability was not observed in [141], where their method was limited to $t = 2$ only and not able to maintain a zero memory overhead for higher error correction capabilities, as shown in Table 12.23. This is primarily due to their inability to accommodate more parity bits. For example, the work in [141] can only accommodate 16-bit parity per 64-bit weight and can correct up to $t = 2$ errors per 64-bit weight with $\sim$ 0.1% storage overhead (which is related to identifying the location of the parity bits during decoding). However, when considering a $t = 3$, they may require an additional $\sim$ 9.4% of memory overhead, which would increase further for $t = 4$.

The detailed analysis of the memory overhead in [141] is as follows: 48 out of the 64 bits can be used as data bits, leaving the remaining 16 bits to store parity bits. A $(70, 48)$ BCH code can be used to perform up to $t = 3$ error correction, which requires 22 parity bits. So, an additional 6 bits of storage are required to accommodate the remaining parity bits, resulting in a memory overhead of $\sim$ 9.4% per 64-bit weight. The same approach is also applicable to higher correction capabilities, such as $t = 4$, incurring a memory overhead of $\sim$ 20.3%.

Additionally, the proposed NN-ECC eliminates the requirement of masking ECC parity bits before inference begins to restore the original weight bits. This is achievable because, in our method, both data and parity bits actively participate in the inference process. Thus, we have access to parity information during inference, which can offer the advantage of performing error detection and correction simultaneously with the inference process. In contrast, existing approaches [138, 139, 140, 141] need to mask the ECC parity bits before inference starts to recover the original weight values. This further limits their method to perform error detection and correction before inference only.

Furthermore, the proposed NN-ECC eliminates the need for additional memory overhead to identify parity locations during the decoding process, a requirement observed in [141], which may be approximately 0.1% to 0.2% of memory overhead. We adhere to the concept of systematic linear block codes, where all the

parity bits are appended together before or after the data bits. This design choice ensures that the proposed method incurs ***zero*** memory overhead while maintaining multi-bit correction capability. Lastly, with the higher correction ability of the proposed NN-ECC, NN resilience against a large number of randomly injected faults is significantly improved, as illustrated in Figures 12.13 and 12.14. This demonstrates its adaptability to a variety of ECC schemes without loss of performance.

Table 12.22.: Factors distinguish the Proposed method from existing works.

Matrices	[138, 139]	[140]	[141]	**Proposed**
Require specific weight distribution	✓	✓	✓	✗
Relies on higher quantization precision	✓	✓	✓	✗
Limited Error Detection & Correction	✓	✓	✓	✗
Require Parity masking before Inference	✓	✓	✓	✗
Require Storage to identify Parity	✗	✗	✓	✗
Accuracy degradation due to encoding	✓	✗	✓	✗

Note: ✗ denotes that a particular constraint is not required. (i.e., advantage) and ✓ indicates that a particular constraint is required. (i.e., bottleneck).

Table 12.23.: Comparative analysis of memory overhead for ECC parity bits.

Method	[139]		[140]		[141]		**Proposed**	
ECC	$(n^{\bar{p}}, k^{\bar{p}})$	MO	$(n^{\bar{p}}, k^{\bar{p}})$	MO	$(n^{\bar{p}}, k^{\bar{p}})$	MO	$(n^{\bar{p}}, k^{\bar{p}})$	MO
Hamming	(72,64)	10.9%	(64,57)	0%	(64,57)	0%	(64,57)	**0%**
BCH (t=2)	(79,64)	21.9%	(71,56)	10.9%	(64,51)	0.1%	(64,51)	**0%**
BCH (t=3)	(86,64)	32.8%	(78,56)	21.9%	(70,48)	9.4%	(64,45)	**0%**
BCH (t=4)	(93,64)	43.7%	(85,56)	32.8%	(77,48)	20.3%	(64,39)	**0%**

Note: MO represents the memory overhead for ECC parity bits.

12.3.4. Section Conclusion

In this section, we proposed NN-ECC, an efficient and generalized method that embeds the ECC parity bits in the NN weight matrix during NN training, completely eliminating the parity bit storage overhead for different ECC schemes in the NN accelerator memories storing NN model parameters. This way, we provide multi-bit error correction guarantees, as required in safety-critical AI applications. We devised a multi-task training approach for the suggested embedding that achieves the same level of inference accuracy as the baseline. The efficacy of the proposed NN-ECC is tested by state-of-the-art codings, such as the Hamming code (single-bit ECC) and the BCH code (multi-bit ECC), to different DNN models on various benchmarks. The results showed that the NN-ECC can significantly improve fault tolerance depending on the selected ECC, without introducing storage overhead. Notably, the proposed NN-ECC is versatile and applicable to diverse NNs and linear block ECC schemes.

Part IV.

Ensuring Continuous Availability of Edge AI Hardware Accelerator

13. Local Approximation-based Continuous Availability

13.1. Problem Statement and Challenges

13.2. Methodology

We propose an online fault tolerance technique to reduce or eliminate downtimes caused by more sophisticated fault mitigation techniques. The key idea of our method is to introduce low-cost block-wise redundancy to the network by providing hardware backups for layers (or groups of layers) that are most sensitive to faults. This is motivated by the fact that online faults have a different impact on the inference accuracy of the network, depending on the layer in which they are localized. Instead of simply making copies of the sensitive layers and using them as backups when the original ones are failing, we use compression methods to approximate the original blocks with smaller models.

13.2.1. Building Local Approximators

To identify the most fault-sensitive parts of the network, we propose topology-specific block-wise fault sensitivity analysis. To estimate the effect of faults on several memristive crossbar arrays, we perform a Monte Carlo fault simulation.

After the fault-sensitive parts of the network are identified, approximators are built offline for each block. In the worst case, several sensitive parts of the NN may fail simultaneously. Therefore, an approximator was built for each sensitive part. To reduce both the hardware and offline training overhead, we propose a single approximator for several sensitive layers of the NNs that are connected sequentially. Consequently, our block-wise approximation led to a smaller number of layers. Our proposed approach applies compression techniques to the original layer(s) of the NN to create block-wise approximators.

We perform a design space search to find the best approximator for each block of a NN topology. Moreover, since the approximators will be "plugged" into the original network during the proposed online fault tolerance, there can be a size mismatch between the approximators and the original blocks of the NNs. If the approximator's output is smaller than that expected by the next layer, the output resolution (height and width) of the approximator must be up-sampled to the shape that the next layer anticipates. Although there are several ways to achieve this, we propose parameter-free up-sampling by "zero padding" the output of the approximators.

13.2.2. Implementation of Approximators

During operation, the health of the crossbar arrays must be constantly monitored to detect faults. In this thesis, several suitable fault detection techniques have been developed, such as [41, 233, 42, 190, 221], which can complement our approach in this Chapter. Also, other existing work, e.g., [37], is also suitable. Additionally, if the runtime monitoring system is able to determine the severity of the faults, then the approximator can be activated based on the severity of the fault. If there are too many faults in a particular block (e.g., the number is above a predefined threshold), an approximator is activated for this block. The approximation remains active until inference accuracy is satisfactorily restored by more sophisticated techniques, by proper fault removal or fault bypassing techniques. The threshold

Table 13.1.: Analyzed neural networks and their respective baseline accuracies.

Architecture	# of layers	# of parameters	Dataset	Baseline acc.
MLP	3	269,826	MNIST	98.2%
VGG-7	7	148,336	Fashion-MNIST	93.3%
ResNet-18	18	11,173,979	CIFAR-10	92.5%

number of faults for a block is defined as the minimal percentage of faulty memristor devices that lead to an unacceptable accuracy reduction. To determine the threshold, a very conservative estimate can be used for high-performance applications, but for other applications, it can be approximated. The overall implementation of our proposed method is depicted in Figure 13.1. Depending on the health of the crossbar arrays, the controller routes the incoming signal to either the normal layers or the approximators. The controller can be implemented with a 2-to-1 MUX, with the fault detection method providing the control signal.

The original network is mapped to conventional memristor-based crossbar arrays. The proposed local approximators are mapped to significantly smaller crossbar arrays. However, since approximators are also mapped to memristor-based crossbar arrays, they are prone to the same faults as well. Thus, we suggest mapping the approximators to more reliable memristors to ensure their fault-free operation. The reliability of the memristor devices can be increased by technology-specific design decisions, e.g., by increasing device dimensions. Also, because our approximators are much smaller than the original network, the cost is minimally increased (see Section 13.3.4). In addition, the memristor cells of the proposed approximators are expected to degrade much later because they are only activated on demand.

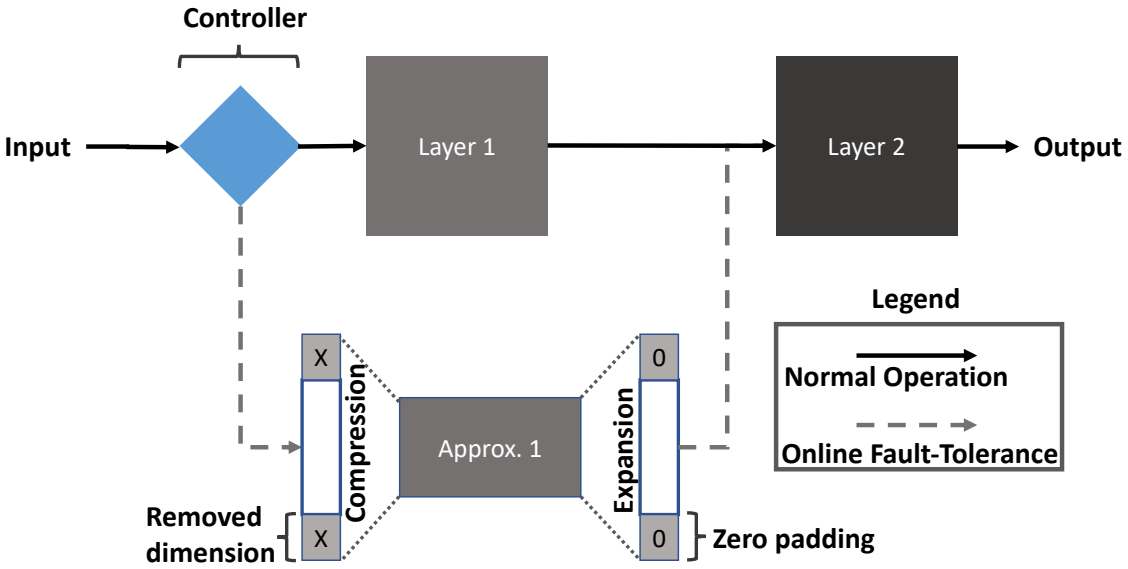


Figure 13.1.: The block diagram of proposed local approximators. Normal layers are active during fault-free operation, whereas local approximators are disabled. Once a sufficient number of faults are detected, the controller activates the compressed local approximators.

13.3. Evaluation

13.3.1. Simulation Setup

Our proposed method has been evaluated on three different topologies and benchmark datasets, as shown in Table 13.1. All networks were trained with 8-bit weights and activations. In this work, faults are modelled as stuck-at-faults (SAFs) as they are the most common type of defects leading to faults in memristive

Table 13.2.: Most fault-sensitive blocks for each topology based on our fault sensitive analysis and approximation criteria.

Topology	Sensitive Layers	Topology	Sensitive Layers
MLP	First layer ($\bar{B}0$)	VGG	First VGG block ($\bar{B}0$)
ResNet-18	First layer ($\bar{B}0$)		Last VGG block ($\bar{B}2$)
	Third residual block ($\bar{B}0$)		All fully connected layers

devices and have a significant impact on inference accuracy. A custom builds PyTorch-based framework was designed to simulate SAFs by randomly (uniform distribution) forcing the faulty bits of the quantized weights to be at a particular state, 0 for stuck-at-0 (SA_0) and 1 for stuck-at-1 (SA_1). Work in [259] shows that about 10% of cells can be affected by SAFs. Additionally, during their lifetime, memory cells wear out and become faulty. As a result, we simulate networks with up to 25% of SAFs, taking samples every 2.5%. For block-wise analysis, the same amounts of SAFs are injected into each block at a time. The sensitive blocks for each topology based on our criteria and analysis is summarized in the Table 13.2.

13.3.2. Constructing of Approximators for Online Fault Tolerance

We analyzed different compression techniques to find the one that is suitable for our requirements. Our goal is to maximize the compression ratio while maintaining inference accuracy sufficiently high. We consider an approximator sufficiently good if the accuracy of the network is within 2.5% of the baseline accuracy when this approximator is used.

Online fault-tolerance for MLP For MLP on the MNIST dataset, when only weights ($\mathbf{W}$) are quantized, the inference accuracy reduces by a negligible amount of 0.2%. Whereas, quantizing both weights and activation ($\mathbf{W}$ & $\mathbf{z}$) causes noticeable accuracy degradation (5%). The results are summarized in Table 13.3. On the other hand, pruning allows removing up to 60% of the neurons without any reduction in accuracy, as summarized in Figure 13.2 (a). Therefore, the sensitive blocks can be quantized as well as pruned. With this approach, 60% of the neurons in the sensitive blocks can be removed while bit-width is reduced to 2-bit. This gives us a compression ratio of 8×. However, the order of pruning and quantization is important. We found that post-quantization pruning is the better way of combining the two techniques. The 2, 3, and 4-bit quantized versions of the layer generally have the same behavior as the original 8-bit layer, as shown in Figure 13.2 (b). With knowledge distillation, we were able to achieve results that are slightly worse than those with post-quantization pruning, though they are comparable. Therefore, post-quantization pruning with an 8.26× compression is used for the approximator block.

Table 13.3.: Inference accuracy of the MLP topology after quantization only weights (W) and weight and activation ($\mathbf{W}$ & $\mathbf{z}$).

Bitwidth	Accuracy (W)	Accuracy (W & z)	Compression ratio
8	98.2% (-0.0%)	98.2% (-0.0%)	**1×**
4	98.5% (+0.3%)	98.4% (+0.2%)	**2×**
3	98.4% (+0.2%)	98.1% (-0.1%)	**2.67×**
2	98.2% (-0.0%)	97.4% (-0.8%)	**4×**
1	98.0% (-0.2%)	93.2% (-5.0%)	**8×**

Online fault-tolerance for VGG Topology For VGG topology $\bar{B}1$ and $\bar{B}2$ blocks are approximated separately, but all the FC layers are approximated into one block. Specifically, the $\bar{B}2$ and FC blocks can be quantized

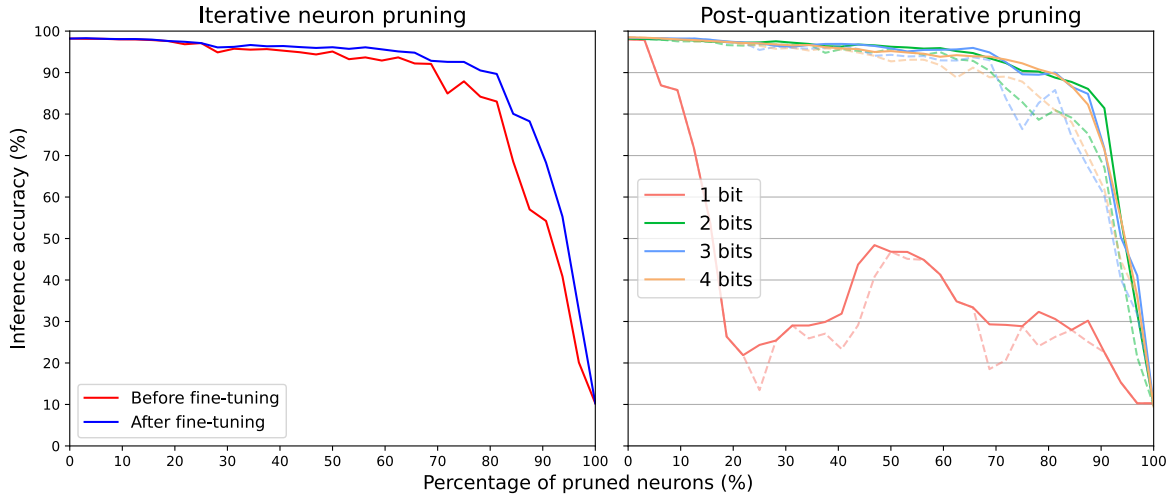


Figure 13.2.: On the left: inference accuracy after pruning MLP. On the right: inference accuracy after quantization and pruning MLP. Dashed lines represent accuracy before fine-tuning. **It is recommended to view this figure in color.**

to 1-bit weights with less than 1% accuracy reduction, but the $\bar{B}1$ block can be quantized to only 3-bits for a similar accuracy reduction. Similarly, with post-quantization pruning, the $\bar{B}2$ block can be compressed significantly more than $\bar{B}1$. We were able to achieve an 8.52× and 8.43× compression for the $\bar{B}1$ and $\bar{B}2$ blocks, respectively, at a similar accuracy (92.3%) to the original network. Since we already had good results with post-quantization pruning, we now apply this method to $\bar{B}1$ and $\bar{B}2$. The fully-connected (FC) block contains a lot of redundant connections. About 90% of neurons can be removed without significant accuracy degradation. Therefore, we try to reduce the number of layers to one with binarized weights. To achieve this, we have used knowledge distillation to compress the FC block by 114.7× with a negligible accuracy degradation of 1.8%. Applying the knowledge distillation method to the $\bar{B}1$ and $\bar{B}2$ blocks can achieve a better compression ratio, but accuracy degrades by ~7% from byseline, which does not meet our requirement.

Online fault-tolerance for ResNet Topology For ResNet topology, the approximators for blocks $\bar{B}0$ and $\bar{B}3$ are designed. Similar to MLP and VGG topology, both the $\bar{B}0$ and $\bar{B}3$ blocks of ResNet can tolerate quantization quite well. Binarization of the weights in the $\bar{B}3$ block does not degrade the inference accuracy at all, whereas $\bar{B}0$ shows an acceptable drop of just 1.6%. Therefore, further compression can be applied with different techniques. We have found that pruning works quite well on both blocks, and about 50% of filters in $\bar{B}0$ and all layers of $\bar{B}3$ can be removed without a big accuracy loss. Post-quantization pruning of a few filters gives us the best result for the $\bar{B}0$ block. We achieve 8.26× compression with an inference accuracy of 91.0% (−1.5%), as summarized in Figure 13.3 (left). However, this method does not work for the $\bar{B}3$ block (see Figure 13.3) (right). Consequently, as an alternative, we first prune 50% of filters in each layer of $\bar{B}3$ and quantize the resulting block to 2 bits, which gives us 90.5% (-2.0%) accuracy at 14× compression.

13.3.3. Multi-Block Fault Tolerance

For VGG topology, when all the approximators are activated at the same time, the accuracy reaches 84.3% (−9%). Table 13.4 shows the accuracy of the network with different combinations of active approximators. Similarly, for ResNet topology, when both approximators are active simultaneously, the inference accuracy reduces to 88.3% (−4.2%). Approximation errors of multiple blocks accumulate to reduce the network's performance, but our method still provides reasonable performance, to achieve graceful degradation.

Table 13.4.: Inference accuracy of the VGG-7 network with different combinations of active approximators

Active approx.	None	$\bar{B}1$	$\bar{B}2$	FC	$\bar{B}1$, $\bar{B}2$	$\bar{B}1$, FC	$\bar{B}2$, FC	$\bar{B}1$, $\bar{B}2$, FC
Accuracy (%)	93.3	93.3	92.3	91.5	89.6	87.6	88.4	84.3

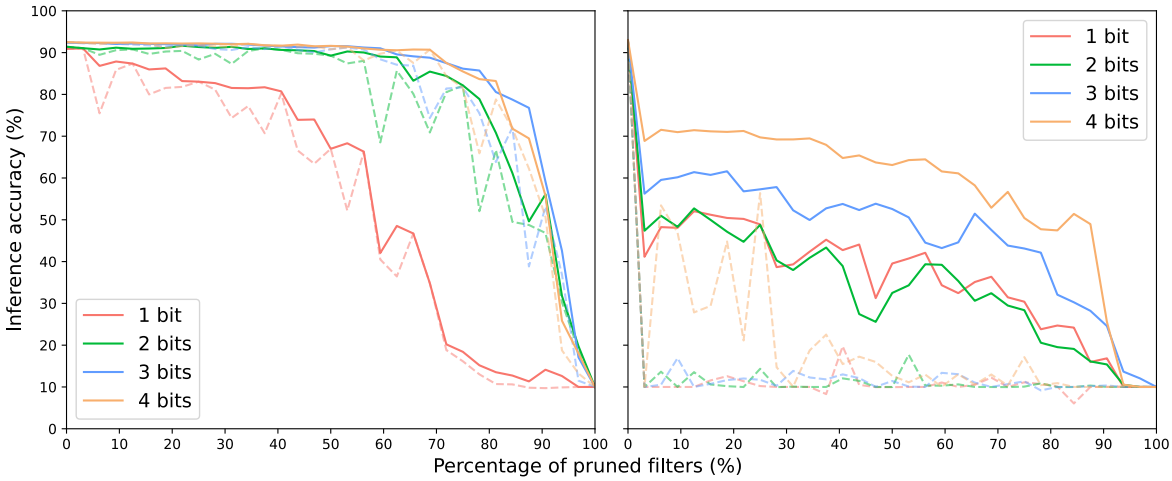


Figure 13.3.: Inference accuracy of ResNet (left) after with post-quatization pruning of block $\bar{B}0$ and (right) after post-quatization of block $\bar{B}3$. Dashed lines represent accuracy before fine-tuning. **It is recommended to view this figure in color.**

13.3.4. Hardware Overhead Analysis

In terms of weight bits, the overhead of the local approximator is only 194.4 Kbit, 67.1 kBit, and 1.18 MBit for the MLP, VGG, and ResNet topology, respectively. Consequently, it results in an overhead of 9.0%, 5.7%, and 1.34%, respectively, compared to the original networks. For comparison, the overhead of a full copy of the original NN with local and global redundancy approaches would be 42.7% and 100%, respectively. Resulting in a reduction in overhead by 33.7% and 98.63%, respectively.

13.4. Scientific Impact of This Work

The major contributions and scientific impacts of this work are as follows:

- **Development of an Online Fault-Tolerance Technique:** Proposed a low-cost, online fault tolerance method based on local approximations to ensure continuous operation of NNs without any downtime. The proposed approach is crucial for 'always-on' applications where system interruptions are unacceptable.
- **Approximate Backups:** The proposed method shows that making the approximation of the sensitive blocks of a NN is an attractive option instead of full redundancy for system availability in a resource constraint system.
- **Scalability:** The proposed approach is also applicable to other AI accelerator architectures such as FPGAs, GPUs, and TPUs. Also, the proposed method is scalable to various NN topologies, including MLPs, CNNs, and more complex structures.

13.5. Chapter Conclusion

In this chapter, we propose an online fault tolerance method for ensuring the availability of neural networks that are implemented on memristive crossbar arrays. Instead of creating copies of the entire network, we propose to protect only the fault-sensitive part of the NNs and apply compression methods to approximate those blocks as backups. In comparison to a conventional global and local redundancy-based approach, our proposed method achieves within 2% accuracy of the original network while reducing hardware overhead by up to 98.63% and 33.7%, respectively.

14. Conclusion and Perspective

In this thesis, we have addressed the critical reliability challenges associated with the deployment of NNs in the resource-constraint edge AI applications and devices. The primary objective was to improve the efficiency of uncertainty estimation, reduce costs of online testing, and improve reliability. Another objective was to offer a low-cost solution for the continuous availability of edge AI hardware while fault detection and maintenance are performed. Our work addresses the critical limitations of existing approaches, which often suffer from significant resource scalability issues, making them impractical for edge devices.

In Part I, we presented several approaches for efficient and scalable uncertainty estimation. Specifically, in Chapter 4, we presented Monte Carlo Dropout-based approaches. We developed the first instance of Dropout-based *binary* Bayesian Neural Network. Our implementation utilizes the stochastic properties of Spintronic devices for Dropout modules, which are integrated into existing memristive crossbar arrays. Later, we introduced the concept of "grouped Dropout," which solves the problem of the integration of binary Bayesian neural networks into convolutional neural networks and significantly reduces the number of Dropout models, power consumption, and chip area. Afterward, we introduced the scale Dropout technique, which requires only a single Dropout module for the entire model, significantly reducing resource overhead, power consumption, and chip area.

In addition, in Chapter 5, we present variational inference-based approaches. In Section 5.1, we propose a memory-centric Bayesian approximation and a novel BayNN topology that allows efficient mapping and on-the-fly sampling from the posterior distribution in CiM architectures. Additionally, we propose a CiM architecture and a CiM aware mapping strategy. Furthermore, in the later work presented in Section 5.2, we propose a novel Bayesian NN framework called "Bayesian subset parameter inference." Specifically, we propose applying Bayesian treatment only to the smallest parameter group, such as the scale vector, while keeping larger parameter groups deterministic. Our approach resulted in the first instance of VI-based BayNN and is implemented in a spintronic-based CiM architecture. Consequently, it reduces memory consumption and sampling time, maintaining resource scalability with minimal random number generators irrespective of model size.

Model Ensemble Approach In Chapter 6, we propose a low-cost yet efficient model ensemble approach where only normalization layers are ensembled, but other parameters are shared among ensemble members. Our approach is grounded by the fact that normalization layers require significantly less storage for parameters and computations. Thus, a better candidate for the ensemble than weights. Consequently, the memory and latency for the inference are significantly reduced compared to related works and are close to the single model. We also propose a hardware accelerator-centric inference method that allows single-shot inference in architecture, such as edge GPUs, using dimension modification, batch-processing, and utilizing the proposed ensemble norm layer. Furthermore, our approach allows updating all ensemble members in a single shot, reducing training costs as well.

In Part II, we presented several approaches for efficient and scalable uncertainty estimation of edge AI accelerator hardware using online functional testing methods. In Chapter 7, we presented several explicit testing methods. We explored an approximate gradient ranking method compaction of test vectors in Section 7.1. Our approach identifies training inputs that are more sensitive to parameter changes based on how much the parameters are modified during training. Our approach can effectively detect hard-to-detect faults with low test queries and significantly minimize testing overhead. Then, in Section 7.2, we also

developed a one-shot testing method, which requires only a single test vector and one forward pass to test an entire model. Our approach relies on detecting distribution shifts in the NN output distribution. A learning algorithm is proposed that generates the proposed one-shot testing test vector. In the later work, presented in Section 7.3, we also proposed a few-shot testing method that can even test a NN with a small number of classes, e.g., binary classifier. Our approach proposes to generate a Bayesian test vector that has its element, e.g., a pixel in an image, represented as a distribution rather than a single value. Therefore, multiple samples can be taken from a single test vector to test the edge AI accelerator.

In Chapter 9, we propose a concurrent testing method for the AI accelerator using a fingerprint-based method. We propose a novel topology with a dual head, with one head giving a normal prediction and the other head giving real-time fingerprints that represent the fault status of the model. By matching the baseline fault-free fingerprint with the real-time fingerprint, faults can be detected concurrently without extra forward passes or test vector storage. Extensive evaluations validated the effectiveness of our method in detecting soft and transient faults.

In Chapter 8, we presented a method for testing Dropout-based BayNNs in Spintronics-implemented CiM architecture. Due to the stochastic output, testing Dropout-based BayNN is difficult and challenging. We investigated the impact of manufacturing and infield non-idealities, affecting different modules of BayNN, on inference accuracy and uncertainty estimates. We propose an automatic test pattern generation method based on the variability ranking of training images and online testing frameworks based on the distribution shift of uncertainty distribution.

Furthermore, in Part III, we proposed several methods for reliability improvement. In Chapter 11, we proposed self-healing approaches for uncertainty reduction. Self-healing is defined as the graceful degradation in accuracy in the presence of faults or variations. In Section 11.1, we propose a quantization algorithm that quantizes each partial sum of a layer in CiM architecture to increase the sensing margin and be tolerant to manufacturing variations. In addition, we propose a design time reference current generation algorithm for the sensing circuits that allows the crossbar arrays in CiM architecture to be tolerant to online thermal variations over the entire operating range of temperature (up to 120 $^{\circ}C$). Later in Section 11.2, we propose a self-healing BayNN without sacrificing the quality of uncertainty estimates. We propose the inverted normalization and affine Dropout concepts that introduce implicit additive and multiplicative noise into the MAC results during training. Also, our normalization layer performs run-time standardization that also aids in fault tolerance via standardizing activations in the case of distribution shift due to faults. Evaluation of our approach shows significant fault tolerance without sacrificing the quality of uncertainty estimates.

In Chapter 12, we proposed approaches for uncertainty reduction via runtime adaptation. Specifically, in Section 12.1, we addressed manufacturing and in-field variations and faults using a low-cost re-calibration method that re-calibrates statistical parameters of normalization layers. Also, to reduce the re-calibration costs, we propose an automatic functional test pattern generation method to compact re-calibration data and approximate batch normalization that reduces computation burden. Afterward, in Section 12.2, we addressed data retention faults and aging-induced drift problems in memristors with approximate scrubbing techniques, retention-aware training algorithms, and weight mapping methods. Our approach proposes to map unstable and stable weights to respective regions of the memristor-based crossbar array. Our proposed learning objective encourages weight organization in the matrix for our scrubbing need. During online operation, unstable weights are frequently scrubbed to restore the respective weight value. Consequently, our approaches can mitigate data retention faults without significant storage overhead, ensuring reliable NN operation for the expected device lifetime. Later, in Section 12.3, we propose a zero-overhead method for guaranteed soft-fault correction in digital AI accelerator architectures. Specifically, we utilize ECC with the target of reducing the overhead of parity information. We propose a multi-task learning algorithm to embed the parity information into the weight matrix during training. During inference, parity information takes part in computations required for a prediction, but during decoding, parity information is used for error correction. Consequently, our approach can maintain the inference accuracy despite single and multiple faults based on the limit of utilized ECC without increasing the parameter count.

Lastly, in Chapter 13, we addressed the problem of long system downtime caused by sophisticated testing and maintenance methods. We propose to make the AI accelerator available during system maintenance by providing low-cost backup. Specifically, instead of making a full copy of the whole system, we only provide backups for most fault-sensitive parts and further compress them via compression methods to reduce costs.

In conclusion, this thesis has made significant strides in addressing uncertainty estimation and reduction challenges for edge AI accelerators deployed in dynamic and uncertain environments via a holistic perspective. We particularly focused on resource-scalable and efficient approaches that are suitable for resource-constrained applications and devices. Our work paves the way for robust and efficient edge AI systems by enhancing uncertainty estimation, testing, and fault tolerance. Consequently, it facilitates the widespread adoption of edge AI in real-world applications in diverse and dynamic environments, including safety-critical applications where reliable predictions are paramount.

14.1. Future Works

This thesis opens several promising directions for future research:

Reducing Uncertainty in Prediction with Continual or Lifelong Learning We mostly focused on reducing uncertainty estimates due to the non-idealities of edge AI accelerators. However, dealing with OoD or uncertainty due to new classes of data in a resource-scalable manner is also important. Therefore, in our future work, we plan to focus on developing continual or lifelong learning techniques to dynamically adapt NNs to new data distributions and reduce uncertainty over time. Therefore, the overall reliability of edge AI accelerators could be further improved in dynamic and uncertain environments.

Zero Overhead ECC for Analog CiM Architectures In the future, we plan on extending our zero-memory overhead ECC method to analog CiM architectures. As a result, this could further enhance the reliability of CiM architectures, ensuring fault tolerance without incurring additional resources. However, the CiM architecture presents some unique challenges, including decoding and keeping the number of columns in the crossbar structure the same. Our goal is to address these challenges.

Reducing Latency of BayNN We have proposed many BayNN approaches that are resource-scalable, power-efficient, and memory-efficient. We also reduce the chip area for BayNN implementations and sampling latency of inference. However, the inference stage still requires $5 - 20$ forward passes. In our future work, we aim to reduce the number of forward passes with one-shot or few-shot Bayesian inference, which translates to a lower latency of BayNN inference.

Fault-tolerance for On-device Training While we proposed several fault-tolerance approaches, from different angles, for on-device inference, on-device AI training has attracted a lot of interest. However, specifically, when an NN is trained on analog CiM architectures, they are susceptible to novel challenges such as reliable weight updates despite write failures, and so on. We also plan to address those challenges and extend some of our approaches for on-device training.

14.2. Perspective

Faults and defects, including permanent and soft faults, affect all computing systems. Therefore, as mentioned repeatedly in the "scientific impact of this work" sections of this thesis, the majority of the works proposed in this thesis are also applicable to other architectures, such as TPUs in edge, FPGAs, edge GPUs, and embedded CPUs.

For example, all the work related to uncertainty estimation presented in Part I can be implemented in other AI accelerator architectures. However, the benefits of CiM and the implementation of Dropout modules using the inherent stochasticity of the device cannot be attained. Furthermore, all the methods for testing proposed in Part II are directly applicable to any edge AI accelerator. Lastly, for reliability improvement, all methods except the self-healing method proposed in Section 11.1 can be applied to improve fault tolerance in other edge AI accelerator architectures.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[2] Joe Lemley, Shabab Bazrafkan and Peter Corcoran. "Deep learning for consumer devices and services: pushing the limits for machine learning, artificial intelligence, and computer vision". In: *IEEE Consumer Electronics Magazine* 6.2 (2017), pp. 48–56.

[3] Shabab Bazrafkan and Peter M Corcoran. "Pushing the AI envelope: merging deep networks to accelerate edge artificial intelligence in consumer electronics devices and systems". In: *IEEE Consumer Electronics Magazine* 7.2 (2018), pp. 55–61.

[4] Nida Shahid, Tim Rappon and Whitney Berta. "Applications of artificial neural networks in health care organizational decision-making: A scoping review". In: *PloS one* 14.2 (2019), e0212356.

[5] Idongesit Zion, Simeon Ozuomba and Philip Asuquo. "An Overview of Neural Network Architectures for Healthcare". In: *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*. IEEE. 2020, pp. 1–8.

[6] J Bughin and J Seong. "Assessing the economic impact of artificial intelligence". In: *ITUTrends Issue Paper* 1 (2018).

[7] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[10] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016).

[11] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[12] Kit Yan Chan, Bilal Abu-Salih, Raneem Qaddoura, Al-Zoubi Ala'M, Vasile Palade, Duc-Son Pham, Javier Del Ser and Khan Muhammad. "Deep neural networks in the cloud: Review, applications, challenges and research directions". In: *Neurocomputing* (2023), p. 126327.

[13] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[14] Wei-Cheng Lin and Yi-Ren Yeh. "Efficient malware classification by binary sequences with one-dimensional convolutional neural networks". In: *Mathematics* 10.4 (2022), p. 608.

[15] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz and William J Dally. "EIE: Efficient inference engine on compressed deep neural network". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 243–254.

[16] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. "In-datacenter performance analysis of a tensor processing unit". In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.

[17] Shimeng Yu. “Neuro-inspired computing with emerging nonvolatile memorys”. In: *Proceedings of the IEEE* 106.2 (2018), pp. 260–285.

[18] Jianmin Chen, Xi Tao, Zhen Yang, Jih-Kwon Peir, Xiaoyuan Li and Shih-Lien Lu. “Guided region-based GPU scheduling: utilizing multi-thread parallelism to hide memory latency”. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE. 2013, pp. 441–451.

[19] T. Y. Lee et al. “World-most energy-efficient MRAM technology for non-volatile RAM applications”. In: *2022 International Electron Devices Meeting (IEDM)*. ISSN: 2156-017X. IEEE, Dec. 2022, pp. 10.7.1–10.7.4. DOI: 10.1109/IEDM45625.2022.10019430.

[20] Dan Hendrycks and Thomas Dietterich. “Benchmarking neural network robustness to common corruptions and perturbations”. In: *arXiv preprint arXiv:1903.12261* (2019).

[21] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016).

[22] Siyu Luan, Zonghua Gu, Leonid B Freidovich, Lili Jiang and Qingling Zhao. “Out-of-distribution detection for deep neural networks with isolation forest and local outlier factor”. In: *IEEE Access* 9 (2021), pp. 132980–132989.

[23] Cesar Torres-Huitzil and Bernard Girau. “Fault and error tolerance in neural networks: A review”. In: *IEEE Access* 5 (2017), pp. 17322–17341.

[24] Hyein Shin, Myeonggu Kang and Lee-Sup Kim. “A thermal-aware optimization framework for ReRAM-based deep neural network acceleration”. In: *Int. Conf. on Computer-Aided Design (ICCAD)*. 2020.

[25] Meiyun Zhang, Shibing Long, Guoming Wang, Yang Li, Xiaoxin Xu, Hongtao Liu, Ruoyu Liu, Ming Wang, Congfei Li, Pengxiao Sun, et al. “An overview of the switching parameter variation of RRAM”. In: *Chinese science bulletin* 59 (2014), pp. 5324–5337.

[26] Liuyang Zhang, Aida Todri-Sanial, Wang Kang, Youguang Zhang, Lionel Torres, Yuanqing Cheng and Weisheng Zhao. “Quantitative evaluation of reliability and performance for STT-MRAM”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2016, pp. 1150–1153.

[27] Christopher Münch, Jongsin Yun, Martin Keim and Mehdi B Tahoori. “Mbist-supported trim adjustment to compensate thermal behavior of mram”. In: *2021 IEEE European Test Symposium (ETS)*. IEEE. 2021, pp. 1–6.

[28] Elbruz Ozen and Alex Orailoglu. “Shaping resilient AI hardware through DNN computational feature exploitation”. In: *IEEE Design & Test* 40.2 (2022), pp. 59–66.

[29] Christopher Münch, Rajendra Bishnoi and Mehdi B Tahoori. “Tolerating Retention Failures in Neuromorphic Fabric based on Emerging Resistive Memories”. In: *2020 25th Asia and South Pacific Design Automat. Conf. (ASP-DAC)*. IEEE. 2020, pp. 393–400.

[30] Vinay Joshi, Manuel Le Gallo, Simon Haefeli, Irem Boybat, Sasidharan Rajalekshmi Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian and Evangelos Eleftheriou. “Accurate deep neural network inference using computational phase-change memory”. In: *Nature communications* 11.1 (2020), p. 2473.

[31] Christopher Münch, Rajendra Bishnoi and Mehdi B Tahoori. “Reliable in-memory neuromorphic computing using spintronics”. In: *Proceedings of the 24th Asia and South Pacific design automation conference*. 2019, pp. 230–236.

[32] Gilbert Sassine, Cécile Nail, Luc Tillie, Diego Alfaro Robayo, Alexandre Levisse, Carlo Cagli, Khalil El Hajjam, Jean-François Nodin, Elisa Vianello, Mathieu Bernard, et al. “Sub-pJ consumption and short latency time in RRAM arrays for high endurance applications”. In: *2018 IEEE International Reliability Physics Symposium (IRPS)*. IEEE. 2018, P–MY.

[33] Meiran Zhao, Huaqiang Wu, Bin Gao, Xiaoyu Sun, Yuyi Liu, Peng Yao, Yue Xi, Xinyi Li, Qingtian Zhang, Kanwen Wang, et al. "Characterizing endurance degradation of incremental switching in analog RRAM for neuromorphic systems". In: *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2018, pp. 20–2.

[34] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu and Daan Wierstra. "Weight Uncertainty in Neural Network". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1613–1622. URL: https://proceedings.mlr.press/v37/blundell15.html.

[35] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[36] Aryan Mobiny, Pengyu Yuan, Supratik K Moulik, Naveen Garg, Carol C Wu and Hien Van Nguyen. "Dropconnect is effective in modeling uncertainty of bayesian deep networks". In: *Scientific reports* (2021).

[37] Ching-Yuan Chen and Krishnendu Chakrabarty. "On-line Functional Testing of Memristor-mapped Deep Neural Networks using Backdoored Checksums". In: *2021 IEEE ITC*.

[38] Research and Markets. *Machine Learning as a Service (MLaaS) Global Market Report 2024*. 2024. URL: https://www.researchandmarkets.com/reports/4806168/machine-learning-as-a-service-mlaas-global (visited on 06/02/2024).

[39] Mengyun Liu and Krishnendu Chakrabarty. "Online fault detection in ReRAM-based computing systems for inferencing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.4 (2022), pp. 392–405.

[40] Hyun-Duk Cho, Ph D Principal Engineer, Kisuk Chung and Taehoon Kim. "Benefits of the big. LITTLE Architecture". In: *EETimes, Feb* (2012).

[41] Soyed Tuhin Ahmed and Mehdi B. Tahoori. "Compact Functional Test Generation for Memristive Deep Learning Implementations using Approximate Gradient Ranking". In: *2022 IEEE International Test Conference (ITC)*. 2022, pp. 239–248. DOI: 10.1109/ITC50671.2022.00032.

[42] Soyed Tuhin Ahmed and Mehdi B Tahoori. "One-shot online testing of deep neural networks based on distribution shift detection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[43] Zangwei Zheng, Mingyuan Ma, Kai Wang, Ziheng Qin, Xiangyu Yue and Yang You. "Preventing Zero-Shot Transfer Degradation in Continual Learning of Vision-Language Models". In: *arXiv preprint arXiv:2303.06628* (2023).

[44] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv and Yoshua Bengio. "Binarized neural networks". In: *NeurIPS* 29 (2016).

[45] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European conference on computer vision*. Springer. 2016, pp. 525–542.

[46] Adrian Bulat and Georgios Tzimiropoulos. "Xnor-net++: Improved binary neural networks". In: *arXiv preprint arXiv:1909.13863* (2019).

[47] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu and Jingkuan Song. "Forward and backward information retention for accurate binary neural networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2250–2259.

[48] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Scale-Dropout: Estimating Uncertainty in Deep Neural Networks Using Stochastic Scale". In: *arXiv preprint arXiv:2311.15816* (2024).

[49] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Scalable Spintronics-based Bayesian Neural Network for Uncertainty Estimation". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2023, pp. 1–6.

[50] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[51] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[52] Dmitry Ulyanov, Andrea Vedaldi and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[53] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[55] Durk P Kingma, Tim Salimans and Max Welling. "Variational dropout and the local reparameterization trick". In: *Advances in neural information processing systems* 28 (2015).

[56] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun and Christoph Bregler. "Efficient object localization using convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 648–656.

[57] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E Turner, Jose Miguel Hernandez-Lobato and Alexander L Gaunt. "Deterministic variational inference for robust bayesian neural networks". In: *arXiv preprint arXiv:1810.03958* (2018).

[58] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi and Kenneth E Goodson. "Phase change memory". In: *Proceedings of the IEEE* 98.12 (2010), pp. 2201–2227.

[59] Hiroyuki Akinaga and Hisashi Shima. "Resistive random access memory (ReRAM) based on metal oxides". In: *Proceedings of the IEEE* 98.12 (2010), pp. 2237–2251.

[60] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, et al. "Spin-transfer torque magnetic random access memory (STT-MRAM)". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 9.2 (2013), pp. 1–35.

[61] Seungchul Jung et al. "A crossbar array of magnetoresistive memory devices for in-memory computing". en. In: *Nature* 601.7892 (2022). ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-021-04196-6.

[62] Tayfun Gokmen, Murat Onen and Wilfried Haensch. "Training deep convolutional neural networks with resistive cross-point devices". In: *Frontiers in neuroscience* 11 (2017), p. 538.

[63] Xiaochen Peng, Rui Liu and Shimeng Yu. "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture". In: *IEEE ISCAS*. IEEE. 2019, pp. 1–5.

[64] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo and Shimeng Yu. "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks". In: *IEEE DATE*. 2018.

[65] Yulhwa Kim, Hyungjun Kim and Jae-Joon Kim. "Neural network-hardware co-design for scalable RRAM-based BNN accelerators". In: *arXiv preprint arXiv:1811.02187* (2018).

[66] Mahta Mayahinia, Abhairaj Singh, Christopher Bengel, Stefan Wiefels, Muath A Lebdeh, Stephan Menzel, Dirk J Wouters, Anteneh Gebregiorgis, Rajendra Bishnoi, Rajiv Joshi, et al. "A voltage-controlled, oscillation-based adc design for computation-in-memory architectures using emerging rerams". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 18.2 (2022), pp. 1–25.

[67] Soyed Tuhin Ahmed, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "Design-time Reference Current Generation for Robust Spintronic-based Neuromorphic Architecture". In: *ACM Journal on Emerging Technologies in Computing Systems* 20.1 (2023).

[68] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang and Joel S Emer. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[69] Sarath Mohanachandran Nair, Christopher Münch and Mehdi B Tahoori. "Defect characterization and test generation for spintronic-based compute-in-memory". In: *2020 IEEE European Test Symposium (ETS)*. IEEE. 2020, pp. 1–6.

[70] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu and Frederick T Chen. "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme". In: *IEEE Transactions on Computers* 64.1 (2014), pp. 180–190.

[71] Manuel Le Gallo and Abu Sebastian. "An overview of phase-change memory device physics". In: *Journal of Physics D: Applied Physics* 53.21 (2020), p. 213002.

[72] Sarath Mohanachandran Nair, Rajendra Bishnoi, Mohammad Saber Golanbari, Fabian Oboril, Fazal Hameed and Mehdi B Tahoori. "VAET-STT: Variation aware STT-MRAM analysis and design space exploration tool". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.7 (2017), pp. 1396–1407.

[73] Rajendra Bishnoi, Mojtaba Ebrahimi, Fabian Oboril and Mehdi B Tahoori. "Read disturb fault detection in STT-MRAM". In: *2014 International Test Conference*. IEEE. 2014, pp. 1–7.

[74] Chuteng Zhou, Prad Kadambi, Matthew Mattina and Paul N Whatmough. "Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation". In: *arXiv preprint arXiv:2001.04974* (2020).

[75] Hyeongsu Kim, Jong-Ho Bae, Suhwan Lim, Sung-Tae Lee, Young-Tak Seo, Dongseok Kwon, Byung-Gook Park and Jong-Ho Lee. "Efficient precise weight tuning protocol considering variation of the synaptic devices and target accuracy". In: *Neurocomputing* 378 (2020), pp. 189–196.

[76] Lizhou Wu, Mottaqiallah Taouil, Siddharth Rao, Erik Jan Marinissen and Said Hamdioui. "Survey on STT-MRAM testing: Failure mechanisms, fault models, and tests". In: *arXiv preprint arXiv:2001.05463* (2020).

[77] J. Park, M. Jo, E. M. Bourim, J. Yoon, D. Seong, J. Lee, W. Lee and H. Hwang. "Investigation of State Stability of Low-Resistance State in Resistive Memory". In: *IEEE Electron Device Letters* 31.5 (2010), pp. 485–487. DOI: 10.1109/LED.2010.2042677.

[78] K. Hofmann, K. Knobloch, C. Peters and R. Allinger. "Comprehensive statistical investigation of STT-MRAM thermal stability". In: *Symposium on VLSI Technology: Digest of Technical Papers*. June 2014, pp. 1–2. DOI: 10.1109/VLSIT.2014.6894367.

[79] K. Tsunoda, M. Aoki, H. Noshiro, Y. Iba, S. Fukuda, C. Yoshida, Y. Yamazaki, A. Takahashi, A. Hatada, M. Nakabayashi, Y. Tsuzaki and T. Sugii. "Area dependence of thermal stability factor in perpendicular STT-MRAM analyzed by bi-directional data flipping model". In: *IEEE International Electron Devices Meeting*. Dec. 2014, pp. 19.3.1–19.3.4.

[80] Zhitao Diao, Zhanjie Li, Shengyuang Wang, Yunfei Ding, Alex Panchula, Eugene Chen, Lien-Chang Wang and Yiming Huai. "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory". In: *Journal of Physics: Condensed Matter* 19.16 (2007), p. 165209.

[81] Helia Naeimi, Charles Augustine, Arijit Raychowdhury, Shih-Lien Lu and James Tschanz. "STTRAM SCALING AND RETENTION FAILURE." In: *Intel Technology Journal* 17.1 (2013).

[82] N. Sayed, S. M. Nair, R. Bishnoi and M. B. Tahoori. "Process variation and temperature aware adaptive scrubbing for retention failures in STT-MRAM". In: *2018 23rd Asia and South Pacific Design Automat. Conf. (ASP-DAC)*. 2018, pp. 203–208. DOI: 10.1109/ASPDAC.2018.8297306.

[83] Nunzio Mirabella, Michelangelo Grosso, Giovanna Franchino, Salvatore Rinaudo, Ioannis Deretzis, Antonino La Magna and M Sonza Reorda. "Comparing different solutions for testing resistive defects in low-power SRAMs". In: *LATS*. IEEE. 2021.

[84] Soyed Tuhin Ahmed, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Anghel Lorena and Mehdi B Tahoori. "Binary bayesian neural networks for efficient uncertainty estimation leveraging inherent stochasticity of spintronic devices". In: *NANOARCH'22: 17th ACM International Symposium on Nanoscale Architectures*. ACM. 2022, pp. 1–6.

[85] Soyed Tuhin Ahmed, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "SpinDrop: Dropout-Based Bayesian Binary Neural Networks With Spintronic Implementation". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 13 (2023). DOI: 10.1109/JETCAS.2023.3242146.

[86] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Spatial-SpinDrop: Spatial Dropout-based Binary Bayesian Neural Network with Spintronics Implementation". In: *IEEE Transactions on Nanotechnology* (2024).

[87] Eiji Fujiwara. *Code Design for Dependable Systems: Theory and Practical Application*. USA: Wiley-Interscience, 2006. ISBN: 0471756180.

[88] Jiaqiang Li, Pedro Reviriego, Liyi Xiao and Haotian Wu. "Protecting Memories against Soft Errors: The Case for Customizable Error Correction Codes". In: *IEEE Transactions on Emerging Topics in Computing* 9.2 (2021), pp. 651–663. DOI: 10.1109/TETC.2019.2953139.

[89] Shu Lin and Daniel J. Costello. *Error control coding: fundamentals and applications*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2004.

[90] R. W. Hamming. "Error detecting and error correcting codes". In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[91] S. Lin and D.J. Costello. *Error Control Coding: Fundamentals and Applications*. Computer applications in electrical engineering series. Prentice-Hall, 1983. ISBN: 9780132837965.

[92] João Filipe Ferreira, Jorge Lobo and Jorge Dias. "Bayesian real-time perception algorithms on GPU". en. In: *Journal of Real-Time Image Processing* 6.3 (Sept. 2011), pp. 171–186. ISSN: 1861-8219. DOI: 10.1007/s11554-010-0156-7. URL: https://doi.org/10.1007/s11554-010-0156-7 (visited on 05/17/2022).

[93] Sara Zermani, Catherine Dezan, Hanen Chenini, Jean-Philippe Diguet and Reinhardt Euler. "FPGA implementation of Bayesian network inference for an embedded diagnosis". In: *2015 IEEE Conference on Prognostics and Health Management (PHM)*. June 2015, pp. 1–10. DOI: 10.1109/ICPHM.2015.7245057.

[94] Glenn G. Ko, Yuji Chai, Marco Donato, Paul N. Whatmough, Thierry Tambe, Rob A. Rutenbar, David Brooks and Gu-Yeon Wei. "A 3mm2 Programmable Bayesian Inference Accelerator for Unsupervised Machine Perception using Parallel Gibbs Sampling in 16nm". In: *2020 IEEE Symposium on VLSI Circuits*. ISSN: 2158-5636. June 2020, pp. 1–2. DOI: 10.1109/VLSICircuits18222.2020.9162784.

[95] Hiromitsu Awano and Masanori Hashimoto. "BYNQNet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1402–1407.

[96] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Shuanglong Liu, Xinyu Niu, Miguel R. D. Rodrigues and Wayne Luk. "FPGA-Based Acceleration for Bayesian Convolutional Neural Networks". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.12 (Dec. 2022). Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 5343–5356. ISSN: 1937-4151. DOI: 10.1109/TCAD.2022.3160948. (Visited on 01/05/2024).

[97] Hongxiang Fan, Martin Ferianc, Miguel Rodrigues, Hongyu Zhou, Xinyu Niu and Wayne Luk. "High-Performance FPGA-based Accelerator for Bayesian Neural Networks". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE Press, Dec. 2021, pp. 1063–1068. DOI: 10.1109/DAC18074.2021.9586137. (Visited on 01/05/2024).

[98] Mark Horowitz. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

[99] Akul Malhotra, Sen Lu, Kezhou Yang and Abhronil Sengupta. "Exploiting Oxide Based Resistive RAM Variability for Bayesian Neural Network Hardware Design". In: *IEEE Transactions on Nanotechnology* 19 (2020). Conference Name: IEEE Transactions on Nanotechnology, pp. 328–331. ISSN: 1941-0085. DOI: 10.1109/TNANO.2020.2982819.

[100] Thomas Dalgaty, Niccolo Castellani, Clément Turck, Kamel-Eddine Harabi, Damien Querlioz and Elisa Vianello. "In situ learning using intrinsic memristor variability via Markov chain Monte Carlo sampling". en. In: *Nature Electronics* 4.2 (Feb. 2021). Number: 2 Publisher: Nature Publishing Group, pp. 151–161. ISSN: 2520-1131. DOI: 10.1038/s41928-020-00523-3. (Visited on 05/17/2022).

[101] Djohan Bonnet, Tifenn Hirtzlin, Atreya Majumdar, Thomas Dalgaty, Eduardo Esmanhotto, Valentina Meli, Niccolò Castellani, Simon Martin, Jean-Francois Nodin, Guillaume Bourgeois, et al. "Bringing uncertainty quantification to the extreme-edge with memristor-based Bayesian neural networks". In: (2023).

[102] Kezhou Yang, Akul Malhotra, Sen Lu and Abhronil Sengupta. "All-spin Bayesian neural networks". In: *IEEE Transactions on Electron Devices* 67.3 (2020), pp. 1340–1347.

[103] Anni Lu et al. "An Algorithm-Hardware Co-Design for Bayesian Neural Network Utilizing SOT-MRAM's Inherent Stochasticity". In: *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 8.1 (June 2022).

[104] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures". In: *ACM Transactions on Embedded Computing Systems* 22.5s (Sept. 2023), 131:1–131:25. ISSN: 1539-9087. DOI: 10.1145/3609116. URL: https://doi.org/10.1145/3609116.

[105] Mattias Teye, Hossein Azizpour and Kevin Smith. "Bayesian uncertainty estimation for batch normalized deep networks". In: *ICML*. PMLR. 2018.

[106] Lars Kai Hansen and Peter Salamon. "Neural network ensembles". In: *IEEE transactions on pattern analysis and machine intelligence* 12.10 (1990), pp. 993–1001.

[107] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[108] David Opitz and Richard Maclin. "Popular ensemble methods: An empirical study". In: *Journal of artificial intelligence research* 11 (1999), pp. 169–198.

[109] Cristian Buciluǎ, Rich Caruana and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.

[110] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[111] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft and Kilian Q Weinberger. "Snapshot ensembles: Train 1, get m for free". In: *arXiv preprint arXiv:1704.00109* (2017).

[112] Ilya Loshchilov and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts". In: *arXiv preprint arXiv:1608.03983* (2016).

[113] Johanna Rock, Tiago Azevedo, René de Jong, Daniel Ruiz-Muñoz and Partha Maji. "On efficient uncertainty estimation for resource-constrained mobile applications". In: *arXiv:2111.09838* (2021).

[114] Yeming Wen, Dustin Tran and Jimmy Ba. "Batchensemble: an alternative approach to efficient ensemble and lifelong learning". In: *arXiv preprint arXiv:2002.06715* (2020).

[115] Soyed Tuhin Ahmed, Michael Hefenbrock and Mehdi B. Tahoori. "Tiny Deep Ensemble: Uncertainty Estimation in Edge AI Accelerators via Ensembling Normalization Layers with Shared Weights". In: *2024 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2024, pp. 1–9.

[116] Peng Liu, Zhiqiang You, Jishun Kuang, Zhipeng Hu, Heng Duan and Weizheng Wang. "Efficient March test algorithm for 1T1R cross-bar with complete fault coverage". In: *Electronics Letters* 52.18 (2016), pp. 1520–1522.

[117] Wen Li, Ying Wang, Huawei Li and Xiaowei Li. "RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime". In: *IEEE ICCD)*. 2019.

[118] Bo Luo, Yu Li, Lingxiao Wei and Qiang Xu. "On functional test generation for deep neural network ips". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1010–1015.

[119] G Gavarini, D Stucchi, A Ruospo, G Boracchi and E Sanchez. "Open-set recognition: an inexpensive strategy to increase dnn reliability". In: *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2022, pp. 1–7.

[120] Annachiara Ruospo, Gabriele Gavarini, Antonio Porsia, Matteo Sonza Reorda, Ernesto Sanchez, Riccardo Mariani, Joseph Aribido, Jyotika Athavale, et al. "Image Test Libraries for the on-line self-test of functional units in GPUs running CNNs". In: *28th IEEE European Test Symposium 2023*. IEEE. 2023.

[121] Shuhang Zhang, Grace Li Zhang, Bing Li, Hai Helen Li and Ulf Schlichtmann. "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects". In: *IEEE Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*. 2020.

[122] Majed Valad Beigi and Gokhan Memik. "Thermal-aware optimizations of ReRAM-based neuromorphic computing systems". In: *Proceedings of the 55th Annual Design Automation Conf.* 2018, pp. 1–6.

[123] Jian Meng, Wonbo Shim, Li Yang, Injune Yeo, Deliang Fan, Shimeng Yu and Jaesun Seo. "Temperature-Resilient RRAM-based In-Memory Computing for DNN Inference". In: *IEEE Micro* (2021).

[124] Iason Giannopoulos, Manuel Le Gallo, Vara Prasad Jonnalagadda, Evangelos Eleftheriou and Abu Sebastian. "Temperature Compensation Schemes for In-Memory Computing using Phase-Change Memory". In: *2020 2nd IEEE Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*. 2020.

[125] Sujan K Gonugondla, Ameya D Patil and Naresh R Shanbhag. "SWIPE: enhancing robustness of ReRAM crossbars for in-memory computing". In: *Int. Conf. on Computer-Aided Design (ICCAD)*. 2020.

[126] Yandong Luo, Xiaochen Peng, Ryan Hatcher, Titash Rakshit, Jorge Kittl, Mark S Rodder, Jae-Sun Seo and Shimeng Yu. "A Variation Robust Inference Engine Based on STT-MRAM with Parallel Read-Out". In: *IEEE Int. Symp. on Circuits and Systems (ISCAS)*. 2020.

[127] Shihui Yin, Xiaoyu Sun, Shimeng Yu and Jae-Sun Seo. "High-throughput in-memory computing for binary deep neural networks with monolithically integrated RRAM and 90-nm CMOS". In: *IEEE Transactions on Electron Devices* 67.10 (2020).

[128] Li-Huang Tsai, Shih-Chieh Chang, Yu-Ting Chen, Jia-Yu Pan, Wei Wei and Da-Cheng Juan. "Robust Processing-In-Memory Neural Networks via Noise-Aware Normalization". In: *arXiv preprint arXiv:2007.03230* (2020).

[129] William Wesley Peterson, Wesley Peterson, Edward J Weldon and Edward J Weldon. "Error-correcting codes". In: (1972).

[130] Vera Pless et al. "FJ MacWilliams and NJA Sloane, The theory of error-correcting codes. I and II". In: *Bulletin of the American Mathematical Society* 84.6 (1978), pp. 1356–1359.

[131] Bruce Jacob, David Wang and Spencer Ng. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[132] Moinuddin K Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J Nair and Onur Mutlu. "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems". In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2015, pp. 427–437.

[133] Prashant J Nair, Dae-Hyun Kim and Moinuddin K Qureshi. "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates". In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), pp. 72–83.

[134] Jamie Liu, Ben Jaiyen, Richard Veras and Onur Mutlu. "RAIDR: Retention-aware intelligent DRAM refresh". In: *ACM SIGARCH Computer Architecture News* 40.3 (2012), pp. 1–12.

[135] A. Das and N. A. Touba. "Selective Checksum based On-line Error Correction for RRAM based Matrix Operations". In: *2020 IEEE 38th VLSI Test Symposium (VTS)*. 2020, pp. 1–6. DOI: 10.1109/VTS48691.2020.9107606.

[136] X. Guo, M. N. Bojnordi, Q. Guo and E. Ipek. "Sanitizer: Mitigating the Impact of Expensive ECC Checks on STT-MRAM Based Main Memories". In: *IEEE Transactions on Computers* 67.6 (2018), pp. 847–860. DOI: 10.1109/TC.2017.2779151.

[137] F. Brosser, E. Milh, V. Geijer and P. Larsson-Edefors. "Assessing scrubbing techniques for Xilinx SRAM-based FPGAs in space applications". In: *Int. Conf. Field-Program. Technol. (FPT)*. 2014, pp. 296–299. DOI: 10.1109/FPT.2014.7082803.

[138] Minghai Qin, Chao Sun and Dejan Vucinic. "Robustness of neural networks against storage media errors". In: *arXiv preprint arXiv:1709.06173* (2017).

[139] Stéphane Burel, Adrian Evans and Lorena Anghel. "Zero-Overhead Protection for CNN Weights". In: *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2021, pp. 1–6. DOI: 10.1109/DFT52944.2021.9568363.

[140] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou and Seung-Hwan Lim. "In-place zero-space memory protection for cnn". In: *Advances in Neural Information Processing Systems* 32 (2019).

[141] Seo-Seok Lee and Joon-Sung Yang. "Value-aware parity insertion ECC for fault-tolerant deep neural network". In: *2022 DATE*. IEEE. 2022, pp. 724–729.

[142] Di Gao, Qingrong Huang, Grace Li Zhang, Xunzhao Yin, Bing Li, Ulf Schlichtmann and Cheng Zhuo. "Bayesian Inference Based Robust Computing on Memristor Crossbar". In: *2021 58th ACM/IEEE DAC*. 2021, pp. 121–126. DOI: 10.1109/DAC18074.2021.9586160.

[143] Nanyang Ye, Jingbiao Mei, Zhicheng Fang, Yuwen Zhang, Ziqing Zhang, Huaying Wu and Xiaoyao Liang. "BayesFT: Bayesian Optimization for Fault Tolerant Neural Network Architecture". In: *58th ACM/IEEE DAC*. 2021, pp. 487–492. DOI: 10.1109/DAC18074.2021.9586115.

[144] Nanyang Ye, Linfeng Cao, Liujia Yang, Ziqing Zhang, Zhicheng Fang, Qinying Gu and Guang-Zhong Yang. "Improving the robustness of analog deep neural networks through a Bayes-optimized noise injection approach". In: *Communications Engineering* 2.1 (2023), p. 25.

[145] Soyed Tuhin Ahmed, Kamal Danouchi, Guillaume Prenat, Lorena Anghel and Mehdi B Tahoori. "Enhancing Reliability of Neural Networks at the Edge: Inverted Normalization with Stochastic Affine Transformations". In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2024, pp. 1–6.

[146] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo Di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. "Equivalent-accuracy accelerated neural-network training using analogue memory". In: *Nature* 558.7708 (2018).

[147] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang and He Qian. "Fully hardware-implemented memristor convolutional neural network". In: *Nature* 577.7792 (2020), pp. 641–646.

[148] Sujan K Gonugondla, Mingu Kang and Naresh R Shanbhag. "A variation-tolerant in-memory machine learning classifier via on-chip training". In: *IEEE Journal of Solid-State Circuits* 53.11 (2018), pp. 3163–3173.

[149] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu and Tingwen Huang. "Vortex: Variation-aware training for memristor X-bar". In: *Proceedings of the 52nd Annual Design Automation Conference*. 2015, pp. 1–6.

[150] Chenchen Liu, Miao Hu, John Paul Strachan and Hai Li. "Rescuing memristor-based neuromorphic design with high defects". In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2017, pp. 1–6.

[151] Shinsei Yoshikiyo, Naoko Misawa, Chihiro Matsui and Ken Takeuchi. "NN Algorithm Aware Alternate Layer Retraining on Computation-in-Memory for Write Variation Compensation of Non-volatile Memories at Edge AI". In: *2023 7th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE. 2023, pp. 1–3.

[152] Shinsei Yoshikiyo, Naoko Misawa, Kasidit Toprasertpong, Shinichi Takagi, Chihiro Matsui and Ken Takeuchi. "Edge Retraining of FeFET LM-GA CiM for Write Variation & Reliability Error Compensation". In: *2022 IEEE International Memory Workshop (IMW)*. 2022, pp. 1–4. DOI: 10.1109/IMW52921.2022.9779255.

[153] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux and Vahid Partovi Nia. "Regularized binary network training". In: *arXiv preprint arXiv:1812.11800* (2018).

[154] T. Kobayashi, K. Nogami, T. Shirotori, Y. Fujimoto and O. Watanabe. "A current-mode latch sense amplifier and a static power saving input buffer for low-power architecture". en. In: *1992 Symposium on VLSI Circuits Digest of Technical Papers*. Seattle, WA, USA: IEEE, 1992, pp. 28–29. ISBN: 978-0-7803-0701-8. DOI: 10.1109/VLSIC.1992.229252. URL: http://ieeexplore.ieee.org/document/229252/ (visited on 03/17/2022).

[155] Yuanzhuo Qu, Bruce F. Cockburn, Zhe Huang, Hao Cai, Yue Zhang, Weisheng Zhao and Jie Han. "Variation-Resilient True Random Number Generators Based on Multiple STT-MTJs". In: *IEEE Transactions on Nanotechnology* 17.6 (Nov. 2018). Conference Name: IEEE Transactions on Nanotechnology, pp. 1270–1281. ISSN: 1941-0085. DOI: 10.1109/TNANO.2018.2873970.

[156] Won Ho Choi, Yang Lv, Jongyeon Kim, Abhishek Deshpande, Gyuseong Kang, Jian-Ping Wang and Chris H. Kim. "A Magnetic Tunnel Junction based True Random Number Generator with conditional perturb and real-time output probability tracking". In: *2014 IEEE International Electron Devices Meeting*. ISSN: 2156-017X. Dec. 2014, pp. 12.5.1–12.5.4. DOI: 10.1109/IEDM.2014.7047039.

[157] Satoshi Oosawa, Takayuki Konishi, Naoya Onizawa and Takahiro Hanyu. "Design of an STT-MTJ based true random number generator using digitally controlled probability-locked loop". In: *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*. June 2015, pp. 1–4. DOI: 10.1109/NEWCAS.2015.7182089.

[158] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal and Damien Querlioz. "Digital Biologically Plausible Implementation of Binarized Neural Networks With Differential Hafnium Oxide Resistive Memory Arrays". en. In: *Frontiers in Neuroscience* 13 (Jan. 2020), p. 1383. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.01383. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.01383/full (visited on 01/11/2022).

[159] Hsiang-Yun Cheng, Christian Hakert, Kuan-Hsun Chen, Yuan-Hao Chang, Jian-Jia Chen, Chia-Lin Yang, Tei-Wei Kuo, et al. "Future computing platform design: A cross-layer design approach". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 312–317.

[160] Xiangyu Dong, Cong Xu, Yuan Xie and Norman P. Jouppi. "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.7 (2012), pp. 994–1007. DOI: 10.1109/TCAD.2012.2185930.

[161] Joes Staal, Michael D Abràmoff, Meindert Niemeijer, Max A Viergever and Bram Van Ginneken. "Ridge-based vessel segmentation in color images of the retina". In: *IEEE transactions on medical imaging* 23.4 (2004), pp. 501–509.

[162] Walid Al-Dhabyani, Mohammed Gomaa, Hussien Khaled and Aly Fahmy. "Dataset of breast ultrasound images". In: *Data in brief* 28 (2020), p. 104863.

[163] *Electron microscopy dataset*. https://www.epfl.ch/labs/cvlab/data/data-em/.

[164] Olaf Ronneberger, Philipp Fischer and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[165] Vijay Badrinarayanan Alex Kendall and Roberto Cipolla. "Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2017, pp. 57.1–57.12.

[166] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan and Serge Belongie. "Feature pyramid networks for object detection". In: *CVPR*. 2017.

[167] John Lazzaro, Sylvie Ryckebusch, Misha Anne Mahowald and Caver A Mead. "Winner-take-all networks of O (n) complexity". In: *Advances in neural information processing systems* 1 (1988).

[168] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram and Yanzhi Wang. "VIBNN: Hardware acceleration of Bayesian neural networks". In: *ACM SIGPLAN Notices* 53.2 (2018), pp. 476–488.

[169] Xiaotao Jia, Jianlei Yang, Runze Liu, Xueyan Wang, Sorin Dan Cotofana and Weisheng Zhao. "Efficient Computation Reduction in Bayesian Neural Networks Through Feature Decomposition and Memorization". In: *IEEE Trans. on Neural Networks and Learning Systems* 32 (Apr. 2021). ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.2987760.

[170] Ruizhou Ding, Ting-Wu Chin, Zeye Liu and Diana Marculescu. "Regularizing activation distribution for training binarized deep networks". In: *Proceedings of the IEEE/CVF CVPR*. 2019, pp. 11408–11417.

[171] Haotong Qin, Xiangguo Zhang, Ruihao Gong, Yifu Ding, Yi Xu and Xianglong Liu. "Distribution-sensitive information retention for accurate binary neural network". In: *International Journal of Computer Vision* (2022), pp. 1–22.

[172] Lu Hou, Quanming Yao and James T Kwok. "Loss-aware binarization of deep networks". In: *arXiv preprint arXiv:1611.01600* (2016).

[173] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan and Kailash Gopalakrishnan. "Pact: Parameterized clipping activation for quantized neural networks". In: *arXiv preprint arXiv:1805.06085* (2018).

[174] Behzad Razavi. "The StrongARM latch [a circuit for all seasons]". In: *IEEE Solid-State Circuits Magazine* 7.2 (2015), pp. 12–17.

[175] Ruizhou Ding, Ting-Wu Chin, Zeye Liu and Diana Marculescu. "Regularizing activation distribution for training binarized deep networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 11408–11417.

[176] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen and Yuheng Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[177] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu and Junjie Yan. "Differentiable soft quantization: Bridging full-precision and low-bit neural networks". In: *Proceedings of the ICCV*. 2019, pp. 4852–4861.

[178] Lu Hou, Quanming Yao and James T Kwok. "Loss-aware Binarization of Deep Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[179] Prabodh Katti, Nicolas Skatchkovsky, Osvaldo Simeone, Bipin Rajendran and Bashir M. Al-Hashimi. "Bayesian Inference on Binary Spiking Networks Leveraging Nanoscale Device Stochasticity". In: *ISCAS*. 2023, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181438.

[180] Seo-Won Lee et al. "Emerging Three-Terminal Magnetic Memory Devices". In: *Proceedings of the IEEE* 104.10 (Oct. 2016). Conference Name: Proceedings of the IEEE, pp. 1831–1843. ISSN: 1558-2256. DOI: 10.1109/JPROC.2016.2543782.

[181] J Ma, C Ge, Y Wang, X An, J Gao, Z Yu and J He. *COVID-19 CT Lung and Infection Segmentation Dataset. Zenodo*. 2020.

[182] Teresa Mendonça, Pedro M Ferreira, Jorge S Marques, André RS Marcal and Jorge Rozeira. "PH 2-A dermoscopic image database for research and benchmarking". In: IEEE. 2013, pp. 5437–5440.

[183] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning". In: (2011).

[184] Adam Coates, Andrew Ng and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[185] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[186] Kamal Danouchi, Guillaume Prenat and Lorena Anghel. "Spin Orbit Torque-based Crossbar Array for Error Resilient Binary Convolutional Neural Network". In: *23RD IEEE LATIN-AMERICAN TEST SYMPOSIUM*. Montevideo, Uruguay, Sept. 2022. (Visited on 09/05/2023).

[187] Balaji Lakshminarayanan, Alexander Pritzel and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *NeurIPS* (2017).

[188] Wenchong He and Zhe Jiang. "A Survey on Uncertainty Quantification Methods for Deep Neural Networks: An Uncertainty Source Perspective". In: *arXiv preprint arXiv:2302.13425* (2023).

[189] D. Ielmini, N. Lepri, P. Mannocci and A. Glukhov. "Status and challenges of in-memory computing for neural accelerators". In: *2022 International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA)*. ISSN: 1930-8868. Apr. 2022, pp. 1–2. DOI: 10.1109/VLSI-TSA54299.2022.9770972. (Visited on 01/10/2024).

[190] Soyed Tuhin Ahmed, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel and Mehdi B. Tahoori. "Testing Spintronics Implemented Monte Carlo Dropout-Based Bayesian Neural Networks". In: *2022 IEEE European Test Symposium (ETS)*. IEEE. 2024, pp. 1–6.

[191] Corey Lammie, Wei Xiang, Bernabé Linares-Barranco and Mostafa Rahimi Azghadi. "MemTorch: An open-source simulation framework for memristive deep learning systems". In: *Neurocomputing* 485 (2022), pp. 124–133.

[192] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen and Yuheng Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: (2016).

[193] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. "Deep Residual Learning for Image Recognition". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 05/30/2023).

[194] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. en. Tech. rep. arXiv:1409.1556. arXiv:1409.1556 [cs]. arXiv, Apr. 2015. URL: http://arxiv.org/abs/1409.1556 (visited on 05/30/2023).

[195] Shimeng Yu. "Neuro-inspired computing with emerging nonvolatile memorys". In: *Proceedings of the IEEE* 106.2 (2018).

[196] J. Doevenspeck, K. Garello, S. Rao, F. Yasin, S. Couet, G. Jayakumar, A. Mallik, S. Cosemans, P. Debacker, D. Verkest, R. Lauwereins, W. Dehaene and G.S. Kar. "Multi-pillar SOT-MRAM for Accurate Analog in-Memory DNN Inference". In: *2021 Symposium on VLSI Technology*. IEEE, 2021, pp. 1–2.

[197] YC Wu, Kevin Garello, W Kim, M Gupta, M Perumkunnil, V Kateel, S Couet, R Carpenter, S Rao, S Van Beek, et al. "Voltage-gate-assisted spin-orbit-torque magnetic random-access memory for high-density and low-power embedded applications". In: *Physical Review Applied* 15.6 (2021), p. 064015.

[198] Debesh Jha, Pia H Smedsrud, Michael A Riegler, Pål Halvorsen, Thomas de Lange, Dag Johansen and Håvard D Johansen. "Kvasir-seg: A segmented polyp dataset". In: *MultiMedia Modeling: 26th International Conference, MMM 2020, Daejeon, South Korea, January 5–8, 2020, Proceedings, Part II 26*. n.d.: Springer, 2020, pp. 451–462.

[199] Gabriel J Brostow, Julien Fauqueur and Roberto Cipolla. "Semantic object classes in video: A high-definition ground truth database". In: *Pattern Recognition Letters* (2009).

[200] Zhongrui Wang, Huaqiang Wu, Geoffrey W Burr, Cheol Seong Hwang, Kang L Wang, Qiangfei Xia and J Joshua Yang. "Resistive switching materials for information processing". In: *Nature Reviews Materials* 5.3 (2020), pp. 173–195.

[201] Song Han, Huizi Mao and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[202] Biresh Kumar Joardar, Janardhan Rao Doppa, Hai Li, Krishnendu Chakrabarty and Partha Pratim Pande. "ReaLPrune: ReRAM Crossbar-Aware Lottery Ticket Pruning for CNNs". In: *IEEE Transactions on Emerging Topics in Computing* (2022).

[203] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[204] Soyed Tuhin Ahmed and Mehdi B Tahoori. "Fault-tolerant Neuromorphic Computing with Functional ATPG for Post-manufacturing Re-calibration". In: *2022 IEEE 40th VLSI Test Symposium (VTS)*. IEEE. 2022, pp. 1–7.

[205] Soyed Tuhin Ahmed and Mehdi B Tahoori. "Fault-tolerant Neuromorphic Computing with Memristors Using Functional ATPG for Efficient Re-calibration". In: *IEEE Design & Test* (2023).

[206] Andrew G Wilson and Pavel Izmailov. "Bayesian deep learning and a probabilistic perspective of generalization". In: *Advances in neural information processing systems* 33 (2020), pp. 4697–4708.

[207] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, et al. "Memristor based computation-in-memory architecture for data-intensive applications". In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2015, pp. 1718–1725.

[208] Shimeng Yu, Hongwu Jiang, Shanshi Huang, Xiaochen Peng and Anni Lu. "Compute-in-memory chips for deep learning: Recent trends and prospects". In: *IEEE circuits and systems magazine* 21.3 (2021), pp. 31–56.

[209] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna and Rachata Ausavarungnirun. "Processing data where it makes sense: Enabling in-memory computation". In: *Microprocessors and Microsystems* 67 (2019), pp. 28–41.

[210] Thorbjörn Posewsky and Daniel Ziener. "Efficient deep neural network acceleration through FPGA-based batch processing". In: *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE. 2016, pp. 1–8.

[211] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh and Jianming Liang. "Unet++: Redesigning skip connections to exploit multiscale features in image segmentation". In: *IEEE transactions on medical imaging* 39.6 (2019), pp. 1856–1867.

[212] José Miguel Hernández-Lobato and Ryan Adams. "Probabilistic backpropagation for scalable learning of bayesian neural networks". In: *International conference on machine learning*. PMLR. 2015, pp. 1861–1869.

[213] Raimund Ubar, Jaan Raik and Heinrich Theodor Vierhaus. *Design and test technology for dependable systems-on-chip*. IGI Global, 2011.

[214] Wei Yang. *pytorch-classification*. URL: `https://github.com/bearpaw/pytorch-classification?tab=readme-ov-file`.

[215] Andrew Ilyas, Logan Engstrom, Anish Athalye and Jessy Lin. "Black-box adversarial attacks with limited queries and information". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2137–2146.

[216] Mateusz Buda, Ashirbani Saha and Maciej A Mazurowski. "Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm". In: *Computers in biology and medicine* 109 (2019), pp. 218–225.

[217] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[218] Gao Huang, Zhuang Liu, Laurens Van Der Maaten and Kilian Q Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[219] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[220] Liang-Chieh Chen, George Papandreou, Florian Schroff and Hartwig Adam. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[221] Soyed Tuhin Ahmed and Mehdi Tahoori. "Few-Shot Testing: Estimating Uncertainty of Memristive Deep Neural Networks Using One Bayesian Test Vector". In: (2024). arXiv: `2405.18894 [cs.LG]`.

[222] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding and Jian Sun. "Repvgg: Making vgg-style convnets great again". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 13733–13742.

[223] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[224] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[225] Jonathan Long, Evan Shelhamer and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[226] Alec Radford, Luke Metz and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[227] Negar Rostamzadeh, Seyedarian Hosseini, Thomas Boquet, Wojciech Stokowiec, Ying Zhang, Christian Jauvin and Chris Pal. *Fashion-Gen: The Generative Fashion Dataset and Challenge*. 2018. arXiv: 1806.08317 [stat.ML].

[228] Yaofo Chen. *PyTorch CIFAR Models*. https://github.com/chenyaofo/pytorch-cifar-models. 2013.

[229] Arjun Chaudhuri, Ching-Yuan Chen, Jonti Talukdar and Krishnendu Chakrabarty. "Functional Test Generation for AI Accelerators using Bayesian Optimization". In: *2023 IEEE 41th VLSI Test Symposium (VTS)*. IEEE. 2023, pp. 1–7.

[230] Minah Lee, Anni Lu, Mandovi Mukherjee, Shimeng Yu and Saibal Mukhopadhyay. "CLUE: Cross-Layer Uncertainty Estimator for Reliable Neural Perception using Processing-in-Memory Accelerators". In: *IJCNN*. 2023.

[231] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. "A survey of uncertainty in deep neural networks". In: *Artificial Intelligence Review* 56.Suppl 1 (2023), pp. 1513–1589.

[232] Fei Su et al. "Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives". In: *IEEE Design & Test* (2023).

[233] Soyed Tuhin Ahmed et al. "Concurrent Self-testing and Uncertainty Estimation of Neural Networks Using Uncertainty Fingerprint". In: *arXiv preprint arXiv:2401.01458* (2024).

[234] Sagar Vaze, Kai Han, Andrea Vedaldi and Andrew Zisserman. "Open-set recognition: A good closed-set classifier is all you need?" In: *arXiv preprint arXiv:2110.06207* (2021).

[235] Qing Dong, Zhehong Wang, Jongyup Lim, Yiqun Zhang, Yi-Chun Shih, Yu-Der Chih, Jonathan Chang, David Blaauw and Dennis Sylvester. "A 1Mb 28nm STT-MRAM with 2.8 ns read access time at 1.2 V VDD using single-cap offset-cancelled sense amplifier and in-situ self-write-termination". In: *2018 IEEE Int. Solid-State Circuits Conf.-(ISSCC)*. IEEE. 2018.

[236] E. M. Boujamaa, S. M. Ali, S. N. Wandji, A. Gourio, S. Pyo, G. Koh, Y. Song, T. Song, J. Kye, J. C. Vial, A. Sowden, M. Rathor and C. Dray. "A 14.7Mb/mm2 28nm FDSOI STT-MRAM with Current Starved Read Path, 52Ω/Sigma Offset Voltage Sense Amplifier and Fully Trimmable CTAT Reference". In: *IEEE Symp. on VLSI Circuits*. 2020. DOI: 10.1109/VLSICircuits18222.2020.9162803.

[237] A. Antonyan, S. Pyo, H. Jung, G. Koh and T. Song. "28-nm 1T-1MTJ 8Mb 64 I/O STT-MRAM with symmetric 3-section reference structure and cross-coupled sensing amplifier". In: *IEEE Int. Symp. on Circuits and Systems (ISCAS)*. 2017. DOI: 10.1109/ISCAS.2017.8050918.

[238] Andrew D Kent and Daniel C Worledge. "A new spin on magnetic memories". In: *Nature nanotechnology* 10.3 (2015), pp. 187–191.

[239] Soyed Tuhin Ahmed, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch and Mehdi B Tahoori. "Process and Runtime Variation Robustness for Spintronic-Based Neuromorphic Fabric". In: *2022 IEEE European Test Symposium (ETS)*. IEEE. 2022, pp. 1–2.

[240] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams and Vivek Srikumar. "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016).

[241] Yoshua Bengio, Nicholas Léonard and Aaron Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*. 2013. arXiv: 1308.3432 [cs.LG].

[242] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[243] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[244] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[245] Fabrice Bernard-Granger, Bernard Dieny, Raphael Fascio and Kotb Jabeur. “SPITT: A magnetic tunnel junction SPICE compact model for STT-MRAM”. In: *Proceedings of the MOS-AK Workshop of the Design, Automation & Test in Europe (DATE)*. 2015.

[246] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv and Yoshua Bengio. “Binarized Neural Networks”. In: 29 (2016).

[247] Y. Shih, C. Lee, Y. Chang, P. Lee, H. Lin, Y. Chen, K. Lin, T. Yeh, H. Yu, H. H. L. Chuang, Y. Chih and J. Chang. “Logic Process Compatible 40-nm 16-Mb, Embedded Perpendicular-MRAM With Hybrid-Resistance Reference, Sub- μ A Sensing Resolution, and 17.5-nS Read Access Time”. In: *IEEE Journal of Solid-State Circuits* 54.4 (Apr. 2019). DOI: 10.1109/JSSC.2018.2889106.

[248] Stela Pudar Hozo, Benjamin Djulbegovic and Iztok Hozo. “Estimating the mean and variance from the median, range, and the size of a sample”. In: *BMC medical research methodology* 5.1 (2005), pp. 1–10.

[249] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[250] Yoshua Bengio, Nicholas Léonard and Aaron Courville. “Estimating or propagating gradients through stochastic neurons for conditional computation”. In: *arXiv preprint arXiv:1308.3432* (2013).

[251] Qing Dong, Zhehong Wang, Jongyup Lim, Yiqun Zhang, Mahmut E Sinangil, Yi-Chun Shih, Yu-Der Chih, Jonathan Chang, David Blaauw and Dennis Sylvester. “A 1-Mb 28-nm 1T1MTJ STT-MRAM with single-cap offset-cancelled sense amplifier and in situ self-write-termination”. In: *IEEE Journal of Solid-State Circuits* 54.1 (2018), pp. 231–239.

[252] Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Ku-Feng Lin, Tien-Wei Chiang, Yuan-Jen Lee, et al. “A Reflow-Capable, Embedded 8Mb STT-MRAM Macro with 9ns Read Access Time in 16nm FinFet Logic CMOS Process”. In: *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2020, pp. 11–4.

[253] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[254] Han Xiao, Kashif Rasul and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[255] Soyed Tuhin Ahmed, Surendra Hemaram and Mehdi B Tahoori. “NN-ECC: Embedding Error Correction Codes in Neural Network Weight Memories using Multi-task Learning”. In: *2024 IEEE 42th VLSI Test Symposium (VTS)*. IEEE. 2024, pp. 1–6.

[256] Michael B. Sullivan, Nirmal R. Saxena, Mike O’Connor, Donghyuk Lee, Paul Racunas, Saurabh Hukerikar, Timothy Tsai, Siva Kumar Sastry Hari and Stephen W. Keckler. “Characterizing and Mitigating Soft Errors in GPU DRAM”. In: *IEEE Micro* 42.4 (2022), pp. 69–77. DOI: 10.1109/MM.2022.3163122.

[257] G. Tshagharyan, G. Harutyunyan, S. Shoukourian and Y. Zorian. “Experimental study on Hamming and Hsiao codes in the context of embedded applications”. In: *2017 IEEE East-West Design & Test Symposium (EWDTS)*. 2017, pp. 1–4. DOI: 10.1109/EWDTS.2017.8110065.

[258] Dmitri Strukov. “The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories”. In: *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*. 2006, pp. 1183–1187. DOI: 10.1109/ACSSC.2006.354942.

[259] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu and Frederick T. Chen. “RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme”. In: *IEEE TC* 64 (2015). DOI: 10.1109/TC.2014.12.

Part V.

Appendix

Soyed Tuhin Ahmed

Curriculum Vitae

Education

2020–2024 **Doctor of Engineering (Dr. Ing.) in Computer Science**, *Karlsruhe Institute of Technology (KIT)*, Karlsruhe, Germany.
Thesis: Scalable and Efficient Approaches to Estimating and Reducing Uncertainty in Edge AI, **Advisor: Prof. Dr. Mehdi B. Tahoori, IEEE Fellow**

2016–2019 **Master of Science in Communications Engineering**, *Technische Universität München (TU München)*, München, Germany.
Focus: Computer Architecture, Design Automation, Embedded Systems, **Advisor: Prof. Dr. Ulf Schlichtmann**

2012–2016 **Bachelor of Science in Electrical and Electronics Engineering**, *American International University Bangladesh (AIUB)*, Dhaka, Bangladesh, Summa Cum Laude distinction.
Focus: Computer Architecture, Electronics, Embedded Systems, **Advisor: Assist. Prof. Saiful Islam Khan**

Research Experience

9/2020–Present **Scientific Researcher**, *Karlsruhe Institute of Technology (KIT)*, Karlsruhe, Germany.
Chair of Dependable Nano Computing (CDNC), Institut für Technische Informatik (ITEC)

- Lead and support research to accelerate AI applications on emerging accelerator architectures
- Designing neural network architecture, optimizing training and inference algorithms to improve efficiency, reliability, testability, and uncertainty estimates of next-gen AI workloads
- Collaborate with other researchers across diverse disciplines, including circuit designers, device physics experts, and DfT experts
- Accomplishments: 1) Improved inference accuracy by up to 90% under unreliable computations, 2) designed testability methods that can achieve 100% fault coverage with a reduction in test vectors by $10000\times$, 3) designed Bayesian neural networks methods that reduced energy consumption up to $100\times$, memory overhead by up to $158.7\times$, and can detect up to 100% out-of-distribution data.

9/2018–3/2019 **Research Internship**, *Robert Bosch GmbH Center for Research and Advance Development*, Robert Bosch Campus 1, 71272 Renningen, Germany.

- Researched and benchmarked RISC-V microprocessor architectures (implemented on FPGA) for internal applications
- Benchmarked ARM Cortex-M microcontroller architectures as baseline
- Compiler optimization for internal applications
- Accomplishments: were able to finish the designated tasks 1 month before the expected time

3/2018–5/2018 **Research Internship**, *Technische Universität München (TU München)*, Chair of Electronic Design Automation, Arcisstr. 21, 80333, Munich, Germany.
Advisor: Prof. Dr.-Ing. habil. Helmut Gräb, IEEE Fellow

- Implemented a cryptographic concept in FPGA and performed functional verification
- Designed and verified finite state machines, random number generators, and layout constraints with VHDL

Engineering Experience

4/2019–12/2019 **Part-time: Embedded Software Engineer**, *Infineon Technologies*, Neubiberg, Germany.
Am Campeon 8b, 85579 Neubiberg, Germany

- Developed a code-coverage tool for microcontrollers to ensure performance and reliability
- Programming languages used: C/C++ , Python, CSS, Shell Scripting

Rintheim – Karlsruhe, Germany
+4917634358645 • *soyed.ahmed@kit.edu*
https://soyedtuhinahmed.github.io

- Testing: PyTest, automated test procedures using Jenkins for continuous integration

Entrepreneur Experience

10/2017–02/2018 **MeinTV (Start-up venture to develop an entertainment box)**, *UnternehmerTUM (Center for Innovation and Business Creation at TUM)*, München, Germany.

- Proposed, implemented, and presented a startup project "MeinTV: Entertainment Box"
- Developed leadership, recruitment, and engineering skills
- Lead research, innovation, and embedded system implementation

Funded Research Projects Worked On

- NeuSPIN – An ANR/DFKI project: NeuSPIN stands for Design of a reliable edge neuromorphic system based on spintronics for Green AI
- NeuroTest – A DFG project: Testing Solutions for Neuromorphic Circuits and Architectures

Teaching and Mentoring

Software Engineering Practice [**Requires curriculum development in each semester**]

- Summer 22: Pytorch-Based Neuromorphic Computing Simulation Tool
- Winter 22/23: Neural Network-based Image Classification System on Heterogeneous Platforms
- Winter 23/24: NeuroShift Dashboard: A Web-based Framework for Monitoring Neural Networks Dynamics
- Summer 24: Developing Android/IOS App for Benchmarking AI Applications in Mobile Devices

Thesis Supervision

1. "Ensuring Reliability and Availability of Neural Networks Mapped to Neuromorphic Crossbar Arrays," Roman Rakhmatullin, Karlsruhe Institute of Technology (KIT)
2. "Developing and Evaluating Stochastic Binarized Activations for Uncertainty Estimation in Binary Neural Networks," Marc Simon Falkenberg, Karlsruhe Institute of Technology (KIT)
3. "Developing Machine Learning-based tool," Aylin Kurumahmut, Karlsruhe Institute of Technology (KIT) and CGI Inc.
4. "One-Shot Uncertainty Estimation Using Population Class," Karlsruhe Institute of Technology (KIT), Hakima Marouan Marouan.

Student Researcher

1. July 23 (one month): Coarse-grained Fault Localization of Deep Neural Networks, Marzieh Malekzadeh Mahani, Karlsruhe Institute of Technology (KIT), studying MSc Remote Sensing and Geoinformatics
2. Summer 24: Evaluation of Uncertainty Estimation of Deep Learning Model Under Data Distribution Shift, Lyes Slimani, Daniel Grévent, Florian Felix Zager, Yan-Keon Elster, Julius Emanuel Weilert, Karlsruhe Institute of Technology (KIT)

Summer Internship Summer 22: Analysis and Implementation of Binary Neural Networks for Hardware Acceleration in FPGAs, Mehdi Toundi, Polytech Montpellier, France

Outreach & Professional Development

Reviewer For

- IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems
- Springer Neural Computing and Applications

Co-Founder Jobs for Bangladeshi People in Germany (online platform for helping people land jobs in Germany)

Awards And Nominations

- Marie Curie Fellowship from University of York, UK, 2022, (did not pursue)
- Richard Newton Young Student Fellowship at the Design Automation Conference 2022
- Summa-Cum-Laude and university gold medal from American International University Bangladesh (AIUB), 2016
- Academic Merit Scholarship from American International University Bangladesh (AIUB), 2013-2015
- EIT Digital Scholarships [merit-based scholarships and are offered to the top applicants], entry at KTH Royal Institute of Technology, Sweden, 2016 (did not pursue)
- Selected paper from VLSI Test Symposium 2022 to appear in the special issue of IEEE Design and Test
- Selected paper from Nanoscale Architectures Symposium (NANOARCH 2022) to appear in the special issue of IEEE Transactions on Nanotechnology
- Three best paper award nominations at premier conferences (IEEE DATE 2024, ACM NANOARCH 2022, IEEE VTS 2022)
- Best paper award of IEEE Journal on Emerging and Selected Topics in Circuits and Systems best paper award nominations (Nominees: Prof. Yiran Chen, Duke University USA, Prof. Partha Pratim Pande, Washington State University USA, and Fei Su, Senior lead architect at Intel)
- Received funding and letter of appreciation for the Bachelor's thesis from "Tech Shop BD," the most popular electronic component seller in Bangladesh
- Finalist of the PhD forum of premier conferences such as Design, Automation, and Test in Europe (DATE) ($2\times$) and Embedded System Week ($1\times$)

List of Scientific Publications (First Author)

Journals

[J1] **Ahmed, Soyed Tuhin**, Michael Hefenbrock, Christopher Münch, and Mehdi B. Tahoori, "Neuroscrub+: Mitigating retention faults using flexible approximate scrubbing in neuromorphic fabric based on resistive memories." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 42, no. 5 (2022): 1490-1503

[J2] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori. "SpinDrop: Dropout-Based Bayesian Binary Neural Networks with spintronic Implementation." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 13, no. 1 (2023): 150-164

[J3] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "Fault-tolerant Neuromorphic Computing with Memristors Using Functional ATPG for Efficient Re-calibration." IEEE Design and Test (2023), [Revised Publication for Top Picks in VTS 2022]

[J4] **Ahmed, Soyed Tuhin**, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch, and Mehdi B. Tahoori. "Design-time Reference Current Generation for Robust Spintronic-based Neuromorphic Architecture." ACM Journal on Emerging Technologies in Computing Systems 20, no. 1 (2023): 1-20,

[J5] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori. "SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures, ESWeek 2023/ACM Transactions on Embedded Computing Systems, 2023

[J6] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Spatial-SpinDrop: Spatial Dropout-based Binary Bayesian Neural Network with Spintronics Implementation", IEEE Transactions on Nanotechnology (TNANO) [Accepted], 2023

[J7] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Scale-dropout: Estimating uncertainty in deep neural networks using stochastic scale", IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I) [Under Review], 2023

[J8] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "One-Shot Online Testing of Deep Neural Networks Based on Distribution Shift Detection", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [Accepted], 2023

[J9] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "Concurrent Self-testing and Uncertainty Estimation of Neural Networks Using Uncertainty Fingerprint", IEEE Transactions on Emerging Topics in Computing, 2024 [Under Review]

Conferences

[C1] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Binary Bayesian Neural Networks for Efficient Uncertainty Estimation Leveraging Inherent Stochasticity of Spintronic Devices", IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), 2022 [Best Paper Candidate]

[C2] **Ahmed, Soyed Tuhin**, Michael Hefenbrock, Christopher Münch, and Mehdi B. Tahoori, "NeuroScrub: Mitigating Retention Failures Using Approximate Scrubbing in Neuromorphic Fabric Based on Resistive Memories", Proceedings of the European Test Symposium (ETS), 2021

[C3] **Ahmed, Soyed Tuhin**, Roman Rakhmatullin, and Mehdi B. Tahoori, "Online Fault-Tolerance for Memristive Neuromorphic Fabric Based on Local Approximation", 28th IEEE European Test Symposium, 2023

[C4] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Scalable Spintronics-based Bayesian Neural Network for Uncertainty Estimation", Design, Automation and Test in Europe Conference, 2023

[C5] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, Compact Functional Test Generation for Memristive Deep Learning Implementations using Approximate Gradient Ranking, IEEE International Test Conference (ITC), 2022

[C6] **Ahmed, Soyed Tuhin**, Mahta Mayahinia, Michael Hefenbrock, Christopher Münch, and Mehdi B. Tahoori, "Process and Runtime Variation Robustness for Spintronic-Based Neuromorphic Fabric", IEEE European Test Symposium (ETS), 2022

[C7] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "Fault-tolerant Neuromorphic Computing with Functional ATPG for Post-manufacturing Re-calibration", 40th VLSI Test Symposium (VTS), 2022 [Best Paper Candidate]

[C8] **Ahmed, Soyed Tuhin**, Hemaram, Surendra, Mehdi B. Tahoori, Embedding Neural Network Parameters with Self Error Correcting Coding using Multi-task Learning, 42th VLSI Test Symposium (VTS), 2024

[C9] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Enhancing Reliability of Neural Networks at the Edge: Inverted Normalization with Stochastic Affine Transformations", Design, Automation and Test in Europe Conference (DATE), 2024 [Accepted, Best Paper Candidate]

[C10] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "Testing Spintronics Implemented Monte Carlo Dropout-Based Bayesian Neural Networks", IEEE European Test Symposium, 2024

[C11] **Ahmed, Soyed Tuhin**, Kamal Danouchi, Michael Hefenbrock, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori, "NeuSpin: Design of a Reliable Edge Neuromorphic System Based on Spintronics for Green AI", 2024, Design, Automation and Test in Europe Conference (DATE), 2024

[C12] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "Estimating Model Uncertainty of Memristive Deep Neural Networks Using One Bayesian Test Vector", 2024 [Under Review]

[C13] **Ahmed, Soyed Tuhin**, and Mehdi B. Tahoori, "Tiny Deep Ensemble: Uncertainty Estimation in Edge AI Accelerators via Ensembling Normalization Layers with Shared Weights", International Conference on Computer-Aided Design (ICCAD), 2024 [Under Review]

List of Scientific Publications (Co-Author)

Conferences

[C1] Hemaram, Surendra, **Ahmed, Soyed Tuhin**, Mahta Mayahinia, Christopher Münch, and Mehdi B. Tahoori, A Low Overhead Checksum Technique for Error Correction in Memristive Crossbar for Deep Learning Applications, IEEE 41st VLSI Test Symposium (VTS), 2023

[C2] Jafari, Atousa, Mahta Mayahinia, **Ahmed, Soyed Tuhin**, Christopher Münch, and Mehdi B. Tahoori, MVSTT: A Multi-Value Computation-in-Memory based on Spin-Transfer Torque Memories, 25th Euromicro Conference on Digital System Design (DSD), 2022

PhD-Forum

[P1] **Ahmed, Soyed Tuhin**, Reliable Memristive Neuromorphic In-Memory Computing: An Algorithm-Hardware Co-Design Approach, in Design, Automation and Test in Europe Conference (DATE), 2023

[P2] **Ahmed, Soyed Tuhin**, Scalable Uncertainty Estimation Approaches in Memristive Deep Learning, in Embedded Systems Week (ESWEEK), 2023

[P3] **Ahmed, Soyed Tuhin**, Scalable and Efficient Methods for Uncertainty Estimation and Reduction in Deep Learning, in Design, Automation and Test in Europe Conference (DATE), 2024

Conference/Workshop Presentation

[T1] "NeuroScrub: Mitigating Retention Failures Using Approximate Scrubbing in Neuromorphic Fabric Based on Resistive Memories", Proceedings of the European Test Symposium (ETS), 2021

[T2] "Binary bayesian neural networks for efficient uncertainty estimation leveraging inherent stochasticity of spintronic devices." In 17th ACM International Symposium on Nanoscale Architectures, 2022

[T3] "Enhancing Reliability of Neural Networks at the Edge: Inverted Normalization with Stochastic Affine Transformations." In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2024

[T4] "Testing Spintronics Implemented Monte Carlo Dropout-Based Bayesian Neural Networks." In IEEE European Test Symposium (ETS), 2024

[T5] "SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures" in International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2023

[T6] "Fault-tolerant Neuromorphic Computing with Functional ATPG for Post-manufacturing Re-calibration", 40th VLSI Test Symposium (VTS), 2022

[T7] "Online Fault-Tolerance for Memristive Neuromorphic Fabric Based on Local Approximation", 28th IEEE European Test Symposium, 2023

[T8] "NeuSpin: Design of a Reliable Edge Neuromorphic System Based on Spintronics for Green AI" In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2024

[T9] "Test Vector Compression for the Functional Testing of AI Accelerators." 4th workshop on AI Hardware: Test, Reliability and Security" (AI-TREATS), 2024

List of Figures

List of Tables

A. List of Abbreviations

CiM Computation-in-memory

CNN Convolutional Neural Network

IFM Input feature map

IF Input feature

MW Moving window

OFM Output feature map

FC Fully connected

FP Floating point

LUT Look-up table

AAC Adder-Accumulator circuit

MAC Multiply-Accumulate

ADC Analog-to-digital converter ()

WWL Write Word-Line

RWL Read Word-Line

SOT Spin-orbit-torque

MTJs Magnetic Tunnel Junction

RNG Random number generator

CMOS Complementary metal-oxide-semiconductor

SOTA State-Of-The-Art

Spice Simulation Program with Integrated Circuit Emphasis

FEOL Front-end-of-line

BEOL Back-end-of-line

TMR Tunnel magnetoresistance

FL Free layer in MTJs

STT Spin-Transfer-Torque

Spintronics-CIM Spintronics-based computation-in-memory

TPR True Negative Rate

FNR False Negative Rate

RRAM Resistive random access memory

NAS Neural architecture search

MUT Model under test

KL Kullback–Leibler

TSMC Taiwan Semiconductor Manufacturing Company Limited

MC Monte Carlo

VI Variational Inference

IoU Intersection over Union

PDK Process design kit

BB-BackProp Bayes by Backpropagation

MSE Mean squared error

OSR Open-set recognition

NLL Negative log-likelihood

BayNN Bayesian Neural Networks

B. List of Symbols

l current layer

Q number of data points

K Kernel shape in a convolutional layer K_H, K_W Height and width of a kernel in a convolutional layer

C_{in} Number of input channels

C_{out} Number of output channel

S Stride in a convolutional layer

N Number of cycles

$\boldsymbol{Q}$ Digital value

$\mathcal{D}$ Dataset

θ Overall parameters

$\hat{\theta}$ Stochastic parameters

$\bar{\theta}$ Overall parameters of NN parity bits embedded into weight matrix

$\bar{W}$ Normalized weights

V Voltage

WL Word-line

ρ Dropout probability

$\top$ Matrix transpose operation

$\phi(.)$ Activation function

$\hat{\mathbf{z}}$ normalized activation value

$\text{BatchNorm}_{\gamma,\beta}(.)$ Batch normalization function

$\odot$ Element-wise multiplication

$\otimes$ Binary convolution

$\mathcal{M}$ Dropout mask

$\boldsymbol{\alpha}$ Scale vector

$\hat{\boldsymbol{\alpha}}$ Dropped scale vector

φ Penalty for scale vector

Δ Thermal stability factor

C Number of output classes

τ_0 Attempt time

τ Model precision in MC Dropout

k_B Boltzmann constant

I Current

t Pulse duration

$Ic0$ Critical current

$\mathcal{T}$ Temperature

$\mathbf{z}$ MAC results of a layer

v Number of counter bits

B Batch size

$\mathbf{Z}$ Batched output

$\boldsymbol{\omega}$ Parameters of variational distribution

$q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ Variational distribution

$\mathcal{S}$ set of values

J Number of samples from the VI distribution

v Number of bits

P Parallel state of MTJ

AP anti-parallel state of MTJ

R_{P} and R_{AP} low and high resistance state of MTJ device, respectively

$\hat{q}_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ Approximate posterior distribution of VI

$\mathcal{D}_{train}$ Training dataset

η Learning rate

$\mathcal{F}(.)$ A NN represented as a function

A_{infer} Inference accuracy of NN

G Conductance of memristor

$\mathcal{V}(.)$ Variation model of memristors

η_0 Noise scale for fault injection

$f(\cdot)$ Permanent fault model

th Threshold

$\bar{\mathbf{x}}$ Synthetically generated test vector

$\mathcal{K}$ Iteration

$\mathcal{P}_{flip}$ Percentage of injected bit-flip faults

$\mathcal{R}$ Fault injection runs

$\mathcal{U}$ Uncertainty fingerprint

l and h Boundary values for fault detection

$\mathcal{S}(.)$ SoftMax function

$\bar{\boldsymbol{\alpha}}$ Approximation of batch normalization parameters

σ_{MRAM} manufacturing variations

Ω Momentum of batch normalization

ψ 32-bit integer variable to keep track of unique activation

s Steps

Me Median

W weights

c Counter variable

$\mathcal{L}$ loss function

p dropout probability

T MC samples

H ECC parity check matrix

n code size of ECC

k data size of ECC

k Length of data bits