Karlsruher Institut für Technologie

# Design and Optimization of Reliable Memory-Centric Computing Architectures

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Ali Nezhadi Khelejani

geb. in Tabriz (Iran)

Tag der mündlichen Prüfung: 13. Juli 2026

| | |
|---|---|
| 1. Referent: | Prof. Dr. Mehdi B. Tahoori,<br>Karlsruher Institut für Technologie (KIT), Karlsruhe |
| 2. Referent: | Prof. Dr. Nima TaheriNejad<br>Universität Heidelberg, Heidelberg |

# Acknowledgements

Praise be to the Lord, who enabled me to be and to do, and who helped me through every struggle.

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Dr. Mehdi B. Tahoori.

I am also sincerely grateful to my wonderful colleagues at the Chair of Dependable Nano Computing (CDNC) for fostering a collaborative and inspiring research environment. I would especially like to acknowledge Mahta Mayahinia and Shanmukha Mangadahalli Siddaramu, with whom I had the pleasure of working closely. Their collaboration, expertise, and friendship made this journey both productive and enjoyable. I wish them continued success, happiness, and prosperity in all their future endeavors.

My sincere thanks also go to Sergej Meschkov, Christopher Keßler, Johannes Reibold, and Vincent Meyers for carefully reviewing this dissertation and for their valuable feedback and continuous support throughout its preparation.

Finally, I would like to extend my heartfelt appreciation to our secretary, Ms. Iris Schröder-Piepka, for her exceptional assistance with administrative matters and paperwork. Her efficiency, kindness, and willingness to help greatly facilitated many aspects of my doctoral studies.

I also want to thank my family, who supported me in every way they could.

Finally, I would like to thank Prof. Dr. Nima TaheriNejad for kindly agreeing to serve as my external examiner, and the entire KIT community and faculty members for their support.

Ali Nezhadi Khelejani
76131 Karlsruhe

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 habe 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, 14. Juli 2026
Ali Nezhadi Khelejani

# Abstract

Modern data-intensive workloads increasingly suffer from the *memory wall*: processor performance and system energy are dominated by the cost of moving data between compute cores and main memory rather than by the cost of arithmetic itself. This limits the benefits of conventional CPU-centric scaling and exacerbates memory bandwidth, latency, and energy bottlenecks. Compute-in-Memory (CiM) architectures address this challenge by performing simple operations close to or within the memory arrays, thereby reducing data movement and alleviating the memory wall while preserving efficiency.

Within this broader context, Non-Volatile Memory (NVM) technologies enable dense, low-leakage storage and in-memory computation due to their inherent resistive nature. As a result, NVM-based CiM (NVM-CiM) architectures are particularly promising for future memory-centric systems. However, these designs face fundamental reliability challenges arising from device variability, circuit-level non-idealities, and the analog nature of many in-memory operations. These effects can significantly degrade application-level correctness and limit the large-scale deployment of NVM-CiM.

This dissertation studies how to design, model, and improve the reliability of NVM-CiM so that they deliver their performance and energy benefits under realistic reliability constraints. It focuses on two classes of memory-centric architectures: ❶ NVM-CiM architectures that perform simple Boolean operations inside the memory arrays, and ❷ NVM-based Content Addressable Memory (NVM-CAM) structures that perform associative search and similarity computation inside memory.

First, we introduce *Sisyphus*, a holistic, reliability-aware, full-system evaluation framework for NVM-CiM architectures, augmented with detailed circuit- and technology-level models. Sisyphus enables quantitative assessment of performance, energy, and reliability for realistic CiM-friendly workloads across a broad range of CPU–CiM configurations and NVM technologies. Using Sisyphus, we show that NVM-CiM can achieve more than 7× latency improvement and 19% power savings over CiM-free baselines, while revealing technology-dependent trade-offs that motivate reliability-aware design.

Second, we propose a cross-layer software–hardware co-design for reliable NVM-CiM. At the circuit level, we replace conventional single-reference sensing with a double-reference scheme that identifies ambiguous sensing decisions. At the architectural and application levels, we associate detection flags with memory words and selectively recompute only suspicious results on the CPU. This double-reference and selective-recompute mechanism reduces the effective application-level decision failure rate below $10^{-12}$, while preserving the performance and energy-efficiency benefits of NVM-CiM with about 1% verification overhead.

Third, we conduct an end-to-end technology-to-application analysis of NVM-based similarity computation using two flavors: scouting-based NVM-CiM and NVM-CAM structures. We combine device models, circuit simulations, hardware-aware architectural extensions, and fault injection to quantify the latency, energy, and reliability trade-offs of both flavors under realistic workloads. Our results show latency improvements of up to 12× and substantial energy savings compared to CPU-only execution.

Finally, we present an algorithm–technology co-optimization methodology for reliable NVM-CAM operation in applications that require exact matching. At design time, we statistically tune the reference voltage to tolerate a limited number of mismatches, ensuring that all true matches are captured even under significant device and sensing variability. At run time, the processor verifies only the resulting candidate matches in software. This reduces the Search Error Rate (SeER) to 0% with only 6% runtime and 4% energy overhead compared to an ideal NVM-CAM, and without additional hardware cost.

Overall, this dissertation advances a cross-layer design methodology for reliable NVM-based memory-centric computing. By integrating technology, circuit, architecture, and application perspectives, it demonstrates that NVM-CiM and NVM-CAM architectures can deliver substantial performance and energy benefits while preserving reliability requirements, thereby enabling their practical, large-scale deployment.

# Zusammenfassung

Moderne datenintensive Workloads leiden zunehmend unter der *Memory Wall*: Prozessorleistung und Systemenergie werden stärker durch die Kosten der Datenbewegung zwischen Rechenkernen und Hauptspeicher als durch die Kosten der arithmetischen Operationen selbst dominiert. Dies begrenzt den Nutzen konventioneller, CPU-zentrierter Skalierung und verschärft Engpässe bei Speicherbandbreite, -latenz und -energie. Compute-in-Memory (CiM)-Architekturen adressieren diese Herausforderung, indem sie einfache Operationen nahe bei oder innerhalb der Speicherzellen ausführen und dadurch die Datenbewegung verringern und die Memory Wall überwinden, ohne die Effizienz zu beeinträchtigen.

In diesem weiteren Kontext ermöglichen Non-Volatile Memory (NVM)-Technologien aufgrund ihrer inhärent resistiven Natur dichte, leistungssparende Speicherung und In-Memory-Berechnungen. NVM-based CiM (NVM-CiM)-Architekturen sind daher besonders vielversprechend für zukünftige speicherzentrierte Systeme. Diese Designs sehen sich jedoch grundlegenden Zuverlässigkeitsherausforderungen gegenüber, die aus Bauteilvariabilität, nichtidealischem Schaltungsverhalten und der analogen Natur vieler In-Memory-Operationen resultieren. Diese Effekte können die Anwendungszuverlässigkeit erheblich beeinträchtigen und die großskalige Einführung von NVM-CiM begrenzen.

Diese Dissertation untersucht, wie sich NVM-CiM so entwerfen, modellieren und in ihrer Zuverlässigkeit verbessern lassen, dass sie ihre Leistungs- und Energievorteile unter realistischen Zuverlässigkeitsanforderungen liefern können. Der Fokus liegt auf zwei Klassen speicherzentrierter Architekturen: ❶ NVM-CiM-Architekturen, die einfache Boolesche Operationen innerhalb der Speicherarrays ausführen, und ❷ NVM-basierte Content Addressable Memory-Strukturen (NVM-CAM), die assoziative Suche und Ähnlichkeitsberechnungen im Speicher durchführen.

Zunächst stellen wir *Sisyphus* vor, ein ganzheitliches, zuverlässigkeitsbewusstes Komplettes-System-Evaluierungsframework für NVM-CiM-Architekturen, ergänzt um detaillierte Schaltungs- und Technologiemodelle. Sisyphus ermöglicht eine quantitative Bewertung von Leistung, Energie und Zuverlässigkeit für realistische CiM-freundliche Workloads über ein breites Spektrum von CPU–CiM-Konfigurationen und NVM-Technologien. Mit Sisyphus zeigen wir, dass NVM-CiM mehr als eine 7×-Verkürzung der Latenz und 19% Energieeinsparung gegenüber CiM-freien Baselines erzielen kann, während technologieabhängige Trade-offs offengelegt werden, die zu zuverlässigkeitsbewusstem Design motivieren.

Zweitens schlagen wir ein Cross-Layer-Software–Hardware-Co-Design für zuverlässige NVM-CiM vor. Auf Schaltungsebene ersetzen wir konventionelles Single-Reference-Sensing durch ein Double-Reference-Schema, das mehrdeutige Entscheidungsfälle beim Sensing identifiziert. Auf Architektur- und Anwendungsebene verknüpfen wir Detektionsflags mit Speicherwörtern und berechnen nur verdächtige Ergebnisse selektiv auf der CPU neu. Dieser Double-Reference- und Selective-Recompute-Mechanismus senkt die effektive anwendungsseitige Fehlentscheidungsrate auf unter $10^{-12}$, während die Leistungs- und Energieeffizienzvorteile von NVM-CiM mit etwa 1% Verifikations-Overhead erhalten bleiben.

Drittens führen wir eine durchgängige Technologie-zu-Anwendung-Analyse von NVM-basierter Ähnlichkeitsberechnung in zwei Varianten durch: Scouting-basierte NVM-CiM-Strukturen und NVM-CAM-Strukturen. Wir kombinieren Bauteilmodelle, Schaltungssimulationen, hardwarebewusste architektonische Erweiterungen und Fault Injection, um die Trade-offs von Latenz, Energie und Zuverlässigkeit beider Varianten unter realistischen Workloads zu quantifizieren. Unsere Ergebnisse zeigen Latenzverbesserungen von bis zu 12× und erhebliche Energieeinsparungen im Vergleich zur reinen CPU-Ausführung.

Abschließend präsentieren wir eine Algorithmus–Technologie-Co-Optimierungsmethodik für zuverlässigen NVM-CAM-Betrieb in Anwendungen, die exakte Übereinstimmung erfordern. Zur Designzeit stimmen wir die Referenzspannung statistisch so ab, dass eine begrenzte Zahl von Mismatches toleriert wird, sodass alle tatsächlichen Treffer selbst unter signifikanter Bauteil- und Sensing-Variabilität erfasst werden. Zur Laufzeit überprüft der Prozessor ausschließlich die resultierenden Kandidatentreffer in Software. Dies reduziert die Search Error Rate (SeER) auf 0% bei lediglich 6% Laufzeit- und 4% Energie-Overhead im Vergleich zu einem idealen NVM-CAM und ohne zusätzlichen Hardwareaufwand.

Insgesamt entwickelt diese Dissertation eine Cross-Layer-Designmethodik für zuverlässiges, NVM-basiertes speicherzentriertes Rechnen weiter. Durch die Integration von Technologie-, Schaltungs-, Architektur- und Anwendungsperspektiven wird gezeigt, dass NVM-CiM- und NVM-CAM-Architekturen erhebliche Leistungs- und Energiegewinne bei Einhaltung von Zuverlässigkeitsanforderungen liefern können und damit ihre praktische, großskalige Implementierung ermöglichen.

# Contents

# List of Own Publications

Below is a list of my publications and co-authored works produced during my three-year tenure as a doctoral candidate at Karlsruher Institut für Technologie (KIT) from 2023 to 2026, categorized into those that are included in this dissertation and those that are not.

## 1. List of Publications Included in this Dissertation


[1] A. Nezhadi, O. Chatzopoulos, M. Mayahinia, G. Papadimitriou, M. Tahoori, and D. Gizopoulos. "Sisyphus: Cross-Layer Efficiency Across NVM Technologies in Compute-in-Memory Architectures". In: *2025 IEEE International Test Conference (ITC)*. 2025, pp. 213–222.

[2] A. Nezhadi, O. Chatzopoulos, D. Gizopoulos, and M. Tahoori. "Trust, but Verify: Reliable Compute-in-Memory via Double-Reference Sensing and Selective Recompute". In: *Accepted at IOLTS 2026 Conference*. 2026.

[3] A. Nezhadi, M. Mayahinia, and M. Tahoori. "Cross-Layer Reliability Evaluation of In-Memory Similarity Computation". In: *2024 IEEE International Test Conference (ITC)*. 2024, pp. 81–85.

[4] A. Nezhadi, S. B. Mamaghani, and M. Tahoori. "Algorithm–Technology Co-Optimization for Reliable NVM-CAM Systems". In: *2026 IEEE 44th VLSI Test Symposium (VTS)*. 2026, pp. 1–7.


## 2. List of Publications not Included in this Dissertation


[5] S. M. Siddaramu, A. Nezhadi, M. Mayahinia, S. Ghasemi, and M. B. Tahoori. "Hardware and Software Co-Design for Optimized Decoding Schemes and Application Mapping in NVM Compute-in-Memory Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3744–3755.

[6] S. M. Siddaramu, A. Nezhadi, M. Mayahinia, S. Ozev, and M. B. Tahoori. "Temporal Reference Scouting Logic for PVT Reliable Logic Computation-in-Memory". In: *2026 IEEE 44th VLSI Test Symposium (VTS)*. 2026, pp. 1–7.

[7] H. Farzaneh, J. P. De Lima, A. Nezhadi Khelejani, A. A. Khan, M. Mayahinia, M. Tahoori, and J. Castrillon. “SHERLOCK: Scheduling Efficient and Reliable Bulk Bitwise Operations in NVMs”. In: *Proceedings of the 61st ACM/IEEE Design Automation Conference*. DAC ’24. Association for Computing Machinery, 2024.

[8] Y.-C. Chen, T. Seidl, N. Hölscher, C. Hakert, M. D. Truong, J.-J. Chen, J. P. C. de Lima, A. A. Khan, J. Castrillon, A. Nezhadi, L. Siddhu, H. Nassar, M. Mayahinia, M. B. Tahoori, J. Henkel, N. Wilbert, S. Wildermann, and J. Teich. *Modeling and Simulating Emerging Memory Technologies: A Tutorial*. 2025. arXiv: 2502.10167.

# Part I.

# Preliminaries

# 1. Introduction

In modern computing systems, the Central Processing Unit (CPU) is primarily responsible for data processing, while other components, such as main memory and caches, are largely dedicated to data storage. This CPU-centric design, commonly referred to as the von Neumann architecture [9], leads to substantial data movement between the CPU and main memory. Such frequent data transfers incur high energy costs and performance overheads [10]—a phenomenon widely known as the *memory wall*. Data movement alone can account for 40% to 60% of the total system energy consumption [11, 12, 13], and accessing main memory consumes 100 to 800 times more energy than performing integer or floating-point operations [12, 14].

Conventional Dynamic Random-Access Memory (DRAM) technology also faces fundamental scalability limitations. Over the past two decades, DRAM capacity has increased by more than two orders of magnitude, while bandwidth has improved by only about one order of magnitude [15]. At the same time, the number of processing cores roughly doubles every two years, whereas DRAM capacity doubles only every three years, causing memory capacity to lag behind by approximately 30% over the same period [16]. A key factor contributing to this imbalance is that aggressive scaling of DRAM cells degrades their reliability, making further technology-node scaling increasingly challenging [17]. To overcome these limitations, recent research has explored both advanced packaging and novel device technologies, including 3D-stacked DRAM (e.g., Hybrid Memory Cube (HMC) [18] and High-Bandwidth Memory (HBM) [19]) [20, 21, 22, 23], as well as memristive Non-Volatile Memories (NVMs) such as Spin-Transfer Torque Magnetic RAM (STT-MRAM) [24, 25], Redox-based RAM (ReRAM) [26], and Phase Change Memory (PCM) [27].

In parallel, to reduce data movement and better balance the energy costs of computation and communication, there has been growing interest in Compute-in-Memory (CiM) architectures that perform certain computations directly within or near the memory arrays [28, 29, 30, 31, 32, 33, 34, 35]. CiM architectures (also known as Processing-in-Memory (PiM) or Near-Memory Computing (NMC)) span a spectrum of architectural realizations and implementation technologies. Proposed architectures range from minimal modifications to the existing memory periphery that enable simple but data-intensive operations (e.g., bulk bitwise operations and bulk data copy primitives [36]) to more general designs that integrate lightweight processing units adjacent to the memory arrays. Figure 1.1 illustrates a representative PiM architecture, based on the commercial *UPMEM* prototype [37, 38].

The remainder of this chapter focuses on two classes of NVM-based in-memory architectures that are central to this dissertation: ❶ NVM-based CiM (NVM-CiM) architectures and ❷ NVM-based CAM (NVM-CAM) architectures, which we treat as application-specific

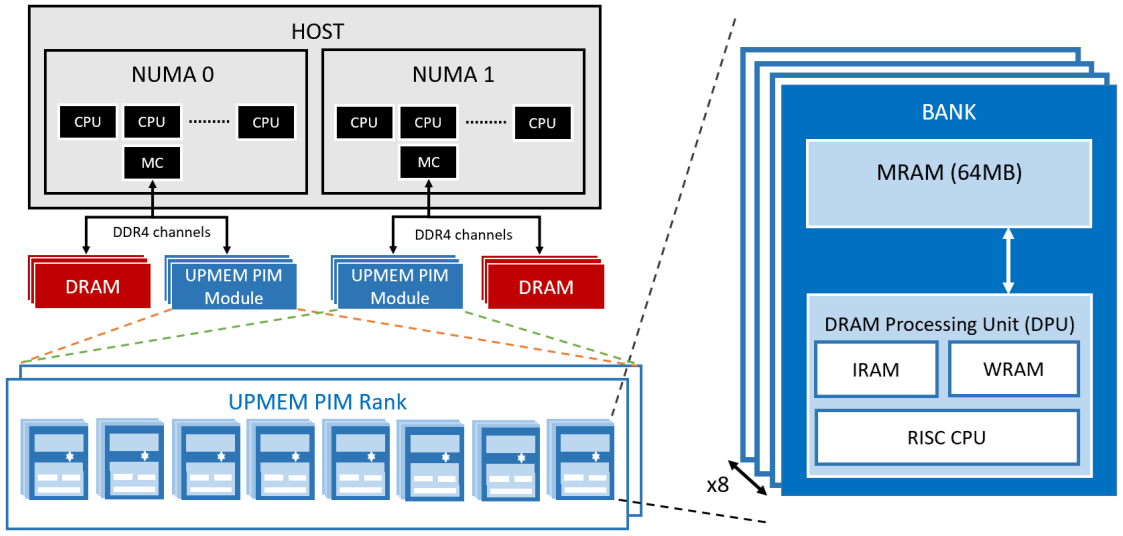



**Figure 1.1.:** Overview of the UPMEM PiM architecture [38]. Lightweight processing units are integrated in the DRAM module periphery, enabling data-parallel CiM operations.

NVM-CiM hardware designs. The following sections provide an overview of these architectures, highlight their advantages, and summarize the reliability challenges that motivate the techniques developed in this dissertation.

## 1.1. NVM-based CiM

Integrating simple processing cores near memory arrays offers the flexibility to execute a broad range of applications. However, such near-memory processor designs typically involve complex microarchitectures and control logic, frequent data exchanges with the CPU, and non-trivial system-level support (e.g., cache coherency mechanisms, shared virtual address spaces, and dedicated programming and task-offloading models) [30, 32, 35]. These challenges limit the scalability and practicality of fully programmable near-memory compute engines.

As a result, a large body of work has increasingly focused on CiM architectures that rely on modest modifications to the memory subarray periphery or bit-cell design [29, 31]. Such designs target specific classes of data-intensive operations and applications, thereby simplifying programming and significantly reducing data transfers with the CPU. In this context, CiM architectures resemble a form of Single Instruction Multiple Data (SIMD) computation but feature much larger register sizes (equivalent to a memory bank row). Many important workloads, including bitmap-indexed databases [39], database search and filtering [40], Deoxyribonucleic Acid (DNA) sequencing and alignment [41, 42], bulk initialization and copy operations [36], and basic image processing kernels [43, 44], can be mapped efficiently to such massively parallel in-memory execution units.

CiM functionality can be realized using various memory technologies, including Static Random-Access Memory (SRAM) [45], DRAM [46, 47], STT-MRAM [48, 49], ReRAM [50, 51], and PCM [52, 53]. Among these, NVM-based designs offer several compelling advantages over conventional DRAM- and SRAM-based computation. First, NVM technologies inherently eliminate static leakage power, as they do not require periodic refresh and enable normally-off, instant-on operation [32, 54]. Second, they generally exhibit more favorable scaling trends than SRAM and DRAM [33, 55], allowing denser integration of memory cells. Third, their resistive nature facilitates analog in-memory computation with modest changes to the read circuitry in crossbar arrays. Examples include implementing Multiply–Accumulates (MACs) in memory crossbars—a key computation in Neural Network (NN) training and inference—and evaluating Boolean functions via multi-row activation [31, 32, 33, 54, 55, 56, 57].

Foundry-developed NVM technologies from TSMC, Samsung, IBM, and Macronix [24, 25, 26, 27] demonstrate that these devices are maturing toward large-scale deployment. Their resistive properties enable energy-efficient analog computation and promise to overcome traditional performance bottlenecks while unlocking unprecedented levels of energy efficiency and throughput. For example, TSMC has recently fabricated ReRAM-based CiM macros targeting NN acceleration [58, 59, 60] and demonstrated STT-MRAM-based CiM modules [61]. Similarly, realistic PCM-based CiM fabrications have been reported in [62], and Samsung and IBM have independently demonstrated STT-MRAM- and PCM-based CiM prototypes [63, 64].

Despite these advances in devices and prototypes, several reliability and design challenges still hinder the practical, large-scale adoption of NVM-CiM. At the device and circuit levels, process variations, device non-idealities, and repeated read/write operations introduce non-negligible error sources. Emerging resistive memories often exhibit relatively high stochastic bit-error rates due to intrinsic device randomness and asymmetric bit-flip probabilities. These effects are further amplified by the analog computation style of many NVM-CiM designs, which makes them particularly sensitive to device-level non-idealities. Consequently, the reliability challenges inherent to NVMs can offset the nominal density and energy-efficiency benefits. At higher fabrication densities, NVMs tend to become increasingly vulnerable to stochastic bit errors [65, 66], requiring more sophisticated protection mechanisms. Therefore, conventional resilience techniques need to be revisited and extended to provide adequate protection in NVM-CiM systems [67, 68, 69]. Moreover, at the array and architectural levels, the type of operation, the number and pattern of operands, and the interaction with peripheral circuitry can significantly affect both correctness and lifetime. Without a rigorous understanding of these cross-layer effects and corresponding mitigation techniques, NVM-CiM systems risk violating application-level reliability and performance targets. In the following section, we build on these observations by examining NVM-CAM architectures, which specialize NVM-based memory arrays for highly parallel associative search.

## 1.2. NVM-based CAM

Modern data-intensive workloads, such as those in cloud services, in-memory databases, and genomic analysis, demand fast and energy-efficient search capabilities across large datasets [70, 71]. Content Addressable Memory (CAM) is a high-speed memory architecture that supports highly parallel, associative data retrieval [72]. Unlike conventional memories, which return the data stored at a specified address, CAM searches its entire contents in a single operation to locate entries that match a query word, offering extremely low-latency lookups.

Traditionally, CAM has been widely used in Translation-Lookaside Buffers (TLBs) for virtual-to-physical address translation and in network routers and switches for packet classification and forwarding [70, 72, 73]. More recently, CAM primitives have been integrated into CiM architectures to accelerate pattern matching [74] and NN classification [75, 76, 77, 78, 79]. Furthermore, CAM functionality is increasingly important in hyperscale and cloud workloads, including networking, databases and key-value stores, genomic analysis, machine learning, and graph analytics [80, 81, 82, 83]. These workloads are often memory-bound and operate on massive datasets.

Popular in-memory database engines such as Redis [84], Memcached [85], Apache Ignite [86], and SQLite (in in-memory mode) [87] are prime candidates for CAM-based acceleration, particularly for workloads dominated by frequent key–value lookups or associative searches.

Genomic analysis is a particularly demanding application domain. Large-scale DNA datasets must be processed using approximate, rather than exact, matching to identify small mutations and sequencing errors. These tasks also require bitwise shift operations, which are inefficient on conventional processors. To meet these demands, researchers have proposed analog and approximate CAM architectures that perform in-memory similarity search to accelerate DNA sequence alignment and comparison [88, 89, 90, 91, 92]. Such designs are typically more complex than standard CAMs because they require advanced matching circuits, multi-bit similarity metrics, and efficient shift logic [93].

Beyond domain-specific accelerators, CAM can also speed up a range of general-purpose data-processing algorithms, such as Locality-Sensitive Hashing [94], Image Similarity Hashing [95], Bloom Filters [96], and bucket sort [97], which are widely deployed in search engines and large-scale cloud platforms.

Despite these advantages, conventional SRAM-based CAMs [70, 71, 72] face significant limitations. Although they deliver high access speeds, each SRAM-based CAM cell typically requires around ten transistors, incurring large area overheads, poor scalability, and limited storage capacity [72]. In addition, high leakage currents and bulky cell structures lead to substantial static power consumption, even when the arrays are idle [98]. These drawbacks hinder the scalability of in-memory SRAM-based CAMs for large-scale, memory-intensive applications [70, 71, 72].

To address these challenges, researchers have explored emerging memory technologies to improve both energy efficiency and density. Among these candidates, NVM technologies again have shown particular promise. By exploiting the unique properties of NVMs, a wide range of NVM-CAM architectures has been proposed to combine fast associative search with improved energy and area efficiency [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122]. Regarding CAM design, NVM technologies offer several key advantages, including high density, low leakage, non-volatility, and compatibility with standard Complementary Metal-Oxide-Semiconductor (CMOS) manufacturing processes, making them strong candidates for next-generation CAM implementations [73, 79, 123]. Some NVM types also provide fast read/write speeds, high endurance, and radiation tolerance, enabling deployment in energy-constrained and harsh environments [124, 125, 126, 127, 128, 129, 130]. Moreover, features such as multi-level storage capability and support for 3D integration enable compact, high-capacity memory arrays [121, 123, 131, 132]. Many NVMs also exhibit inherently resistive storage, where information is encoded in distinct resistance states. This property naturally lends itself to CiM-style operation, allowing data to be processed directly within the memory array using current-mode analog computations [133, 134].

However, similar to NVM-CiM, NVM-CAMs face critical reliability challenges. Device-level variability and non-idealities arising from manufacturing, environmental conditions, and aging, combined with the analog nature of similarity computations using resistive NVMs, lead to deviations in bit-cell resistances from their ideal low- and high-resistance states (low-resistive state (LRS) and high-resistive state (HRS)). In practice, these resistance distributions are well modeled as normal (Gaussian) distributions, which cause non-negligible sensing errors and long-term reliability concerns [75, 98, 124, 125, 126, 131, 132, 133, 134, 135, 136, 137]. Understanding and mitigating these reliability issues is essential for scaling NVM-CAM architectures to large, energy-efficient, and robust in-memory search systems. These reliability challenges in both NVM-CiM and NVM-CAM architectures motivate the cross-layer techniques developed in this dissertation, which are summarized next.

## 1.3. Contributions

This dissertation addresses the aforementioned challenges and advances the design and optimization of reliable memory-centric computing systems based on NVM technologies. The ultimate goal is to enable the practical, large-scale deployment of NVM-CiM and NVM-CAM systems under realistic constraints and without relying on overly idealized assumptions. The dissertation is organized as follows.

**Chapter 2: Background** provides the necessary background on memory-centric computing and NVM technologies. This chapter establishes the terminology and concepts used throughout the remainder of the dissertation. It reviews conventional DRAM- and SRAM-based systems, introduces CiM and CAM architectures, and summarizes key device-, circuit-, and architecture-level characteristics of NVM-based designs.

Finally, it details the reliability challenges related to NVM-CiM and NVM-CAM systems.

**Chapter 3: Benchmarking NVM-CiM Workloads** identifies the lack of a comprehensive, reliability-aware, full-system simulation and evaluation framework for NVM-CiM architectures. To fill this gap, we introduce a cross-layer, full-system simulation framework that enables fast design-space exploration of NVM-CiM systems. Our framework integrates detailed circuit-level modeling of multiple NVM technologies with real-world, data-intensive, CiM-friendly workloads, allowing holistic evaluation of performance, energy, and reliability across a broad range of CPU–CiM configurations. Using this framework, we demonstrate the advantages and limitations of NVM-CiM architectures compared to conventional computing systems.

**Chapter 4: Cross-layer Solution for Reliable NVM-CiM Realization** addresses reliability challenges in NVM-CiM architectures through a software–hardware co-design approach. We introduce double-reference sensing to detect potential errors during CiM execution and a lightweight error-flag mechanism that selectively triggers CPU recomputation for uncertain outputs. Furthermore, we develop an optimization model for reference placement that maximizes reliability while minimizing CPU recomputation, thereby preserving the performance and energy benefits of NVM-CiM.

**Chapter 5: Benchmarking Various NVM-CAM Designs** focuses on NVM-CAM structures for *approximate* in-memory similarity search. We systematically benchmark a range of digital and analog sensing schemes under realistic workloads and quantify their trade-offs in terms of performance, energy, and reliability. Our analysis highlights how different NVM-CAM designs behave in approximate computing contexts and clarifies the conditions under which each design is most suitable.

**Chapter 6: Cross-layer Solution for Reliable NVM-CAM Realization** investigates *exact-match* NVM-CAM structures and addresses their reliability challenges using a cross-layer methodology. We develop protection and mitigation techniques that account for device-level variability, circuit-level non-idealities, and architectural design choices. We demonstrate the benefits of these techniques for representative workloads such as database search and genomic analysis, showing that robust NVM-CAM designs can deliver high performance and energy efficiency while meeting reliability requirements.

**Chapter 7: Conclusion** summarizes the main findings of this dissertation and discusses open challenges and promising directions for future research in reliable NVM-based memory-centric computing.

Taken together, these chapters develop a cross-layer view of reliable NVM-based memory-centric computing, connecting device-, circuit-, architecture-, and application-level insights into a coherent design methodology that guides the remainder of this dissertation.

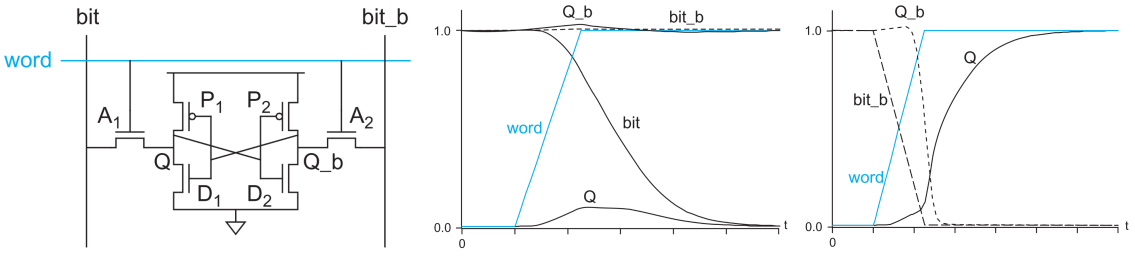
# 2. Background

This chapter reviews key memory technologies and the architectural concepts built upon them. We first summarize conventional charge-based memories, namely SRAM and DRAM, and then introduce emerging NVMs that enable dense storage and computation within memory arrays. Next, we outline CiM paradigms and CAM organizations that leverage these memory technologies to realize energy-efficient, data-centric processing. Finally, we discuss the related reliability challenges associated with NVM-based CiM and CAM systems.

## 2.1. Charge-based Memory Technologies

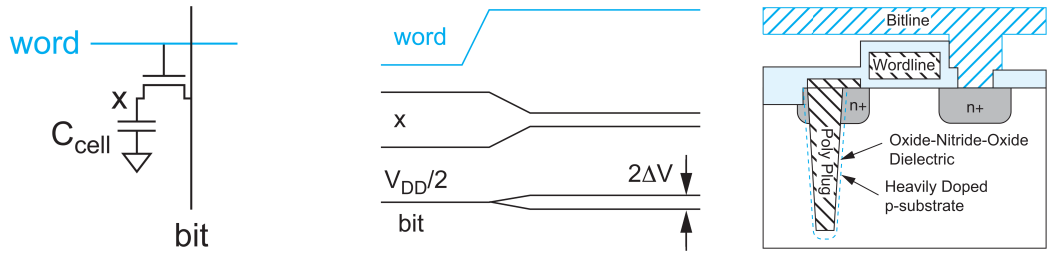
### 2.1.1. SRAM

SRAM is a charge-based volatile memory implemented entirely with CMOS transistors. A typical SRAM bit-cell is built from a pair of cross-coupled inverters plus access transistors (commonly a 6T cell), storing data as a bistable state (high/low voltages). Because no explicit capacitor is used, the stored state is maintained as long as power is applied and the node charge is preserved; no refresh is required. SRAM provides sub-nanosecond access times (<1 ns) and exhibits very high endurance (>$10^{16}$ cycles). The high access speed and robustness make SRAM the standard choice for on-chip caches and for high-performance logic primitives. However, SRAM's multi-transistor cell leads to a relatively large areal footprint, limiting the maximum on-chip capacity even at advanced technology nodes. This is an important trade-off when comparing SRAM with other memory technologies [138]. Figure 2.1 shows the standard 6T SRAM cell and its read (Figure 2.1b) and write (Figure 2.1c) operations.




**(a)** Standard 6T SRAM cell **(b)** Read operation **(c)** Write operation

**Figure 2.1.:** Standard 6T SRAM cell and its read and write operations.




**(a)** Standard 1T1C DRAM cell **(b)** Read operation **(c)** Die schematic and trench capacitor

**Figure 2.2.:** Standard 1T1C DRAM cell, its read operation, and the die schematic of a single cell with its trench capacitor.

### 2.1.2. DRAM

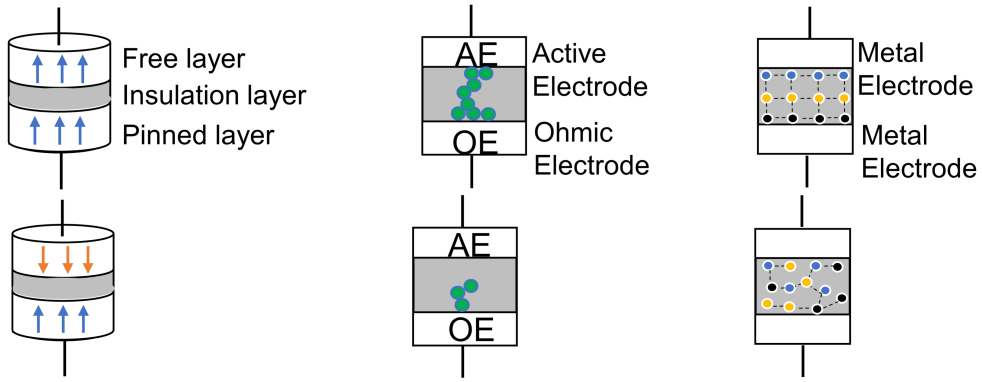
DRAM is a charge-based volatile memory that stores data in explicit capacitor structures, typically implemented with one access transistor and one capacitor per bit-cell (1T1C). Each DRAM cell holds information as the presence or absence of charge on its capacitor, corresponding to logic high and low levels, which are sensed through a highly sensitive Sense Amplifier (SA) during read operations. Because the stored charge gradually leaks away due to parasitic paths, DRAM inherently requires periodic refresh operations to restore the cell contents, even when power remains applied. DRAM offers moderate access times on the order of a few tens of nanoseconds and very high endurance (>$10^{15}$ cycles), but with lower speed than SRAM and additional latency/energy overhead due to refresh and row-activation operations. The compact 1T1C cell, however, enables very high bit densities and significantly lower cost per bit, which makes DRAM the dominant technology for off-chip main memory and large-capacity buffers in modern computing systems [138]. Figure 2.2 shows a standard 1T1C DRAM cell and its read operation (Figure 2.2b). The write operation is similar to the read operation, but in the opposite direction. Additionally, Figure 2.2c shows the die schematic of a single cell and its trench capacitor.

## 2.2. Resistive-based Memory Technologies

Resistive memory technologies store information in the conductance state of a material rather than in the amount of electric charge held on a capacitor or stored in a bistable latch. Bits are encoded as distinct resistance levels, typically at least two resistive states, *low* and *high*, namely LRS and HRS, respectively. These devices can inherently provide non-volatility, enabling data retention without power while still offering access times that approach those of conventional charge-based memories. Furthermore, many resistive memories are compatible with back-end-of-line integration and can be realized in highly scalable, dense crossbar arrays, making them attractive candidates for future main and storage-class memories as well as for in-memory computing primitives. In the following, we focus on three prominent technologies—STT-MRAM, ReRAM, and PCM—and highlight their device-level physics.

**Table 2.1.:** Quantitative comparison of memory technologies [73, 79, 123].

| | SRAM | DRAM | STT-MRAM | ReRAM | PCM |
|---|---|---|---|---|---|
| Cell Size ($F^2$) | 120–200 | 60–100 | 6–50 | 4–10 | 4–12 |
| Write Endurance | $>10^{16}$ | $>10^{15}$ | $10^{12}$–$10^{15}$ | $10^{8}$–$10^{11}$ | $10^{8}$–$10^{9}$ |
| Read Latency (ns) | ~0.2–2 | ~10 | 2–35 | ~10 | 20–60 |
| Write Latency (ns) | ~0.2–2 | ~10 | 3–50 | ~50 | 20–150 |
| Leakage Power | High | Mid | Low | Low | Low |
| Dynamic Energy (Read/Write) | Low | Mid | Low/High | Low/High | Mid/High |
| Commercially Available Size | any | any | ⩽1Gbit | ⩽8Mbit | ⩽128Mbit |




**(a)** STT-MRAM **(b)** ReRAM **(c)** PCM

**Figure 2.3.:** Schematic representations of NVM technologies with LRS on top and HRS on the bottom.

Additionally, Table 2.1 provides a quantitative comparison of the memory technologies discussed. In resistive memories, a read operation senses the cell resistance by applying a voltage to the device and measuring the resulting current. Thus, despite the different physical phenomena that govern NVM technologies, the NVM read procedure is based on resistance sensing and does not differ fundamentally across technologies. However, NVM technologies differ mainly in the write procedure. The write operation in STT-MRAM is considerably less costly in terms of energy and latency than the corresponding operations in ReRAM and PCM.

### 2.2.1. STT-MRAM

The main storage element of STT-MRAM is the Magnetic Tunnel Junction (MTJ), which consists of three layers (Figure 2.3a). Two ferromagnetic layers sandwich an insulating layer. The magnetic orientation of the *free* layer can be rotated, while the magnetic orientation of the *pinned* layer is fixed. In the MTJ, the relative magnetic orientation of the free and pinned layers determines the resistive state. In the case of parallel magnetic orientation, the device is in the LRS, and in the case of antiparallel magnetic orientation, the device is in the HRS. To perform the write operation, a current needs to pass through

the MTJ device. Write current flowing from the free layer to the pinned layer aligns the magnetic orientations of the free and pinned layers in parallel (i.e., programs the device into the LRS), whereas current in the opposite direction programs the device into the HRS [139, 140]. The works in [24, 25] present two examples of STT-MRAM chips by TSMC and Samsung, respectively.

### 2.2.2. ReRAM

As shown in Figure 2.3b, the ReRAM cell is also a stack of three layers. Two metallic (*ohmic* and *active*) electrodes sandwich an oxide-based insulating layer. Within the oxide-based insulating layer, the oxygen vacancies can form a conducting filament connecting the metal electrodes. If a conducting filament exists in the insulating layer, the device is in LRS; otherwise, it is in HRS. To perform the write operation, a voltage needs to be applied to the ReRAM device. Since the work functions of the ohmic and active electrodes are not equal, a certain voltage polarity shapes the filament (*SET*) and programs the device into LRS. Conversely, reversing the voltage polarity ruptures the filament (*RESET*) and programs the device into HRS [141, 142]. ReRAM technology has been successfully demonstrated by TSMC [26].

### 2.2.3. PCM

Similarly, as shown in Figure 2.3c, PCM cells also consist of three layers: two metallic electrodes and, between them, a phase-change material. The phase-change material can be either in the *crystalline* or *amorphous* phase, which results in the LRS and HRS, respectively. To perform the write operation, an electric pulse is applied to the PCM device. By applying a long-duration, low-amplitude pulse, the phase-change material can be heated below the melting temperature, which induces the amorphous-to-crystalline phase change, i.e., HRS-to-LRS switching. For the reverse switching, the phase-change material needs to be heated above the melting temperature and then immediately cooled down. To fulfill this action, a short-duration, high-amplitude electric pulse is required [143, 144]. IBM and Macronix have already demonstrated successful PCM fabrication [27].

## 2.3. NVM-CiM: Stateful vs. Stateless Paradigms

CiM architectures leverage memory arrays not only for data storage but also as active computational substrates, thereby alleviating the energy and latency overheads associated with frequent data movement between separate memory and processing units [28, 145]. In the context of resistive memories such as STT-MRAM, ReRAM, and PCM, CiM builds on the memristive behavior of resistive switching devices [55, 146, 147]. Hence, two broad paradigms for logic and arithmetic emerge: ❶ *stateful* CiM, in which the logical state is represented and updated directly in the device resistance states, and ❷ *stateless* (or

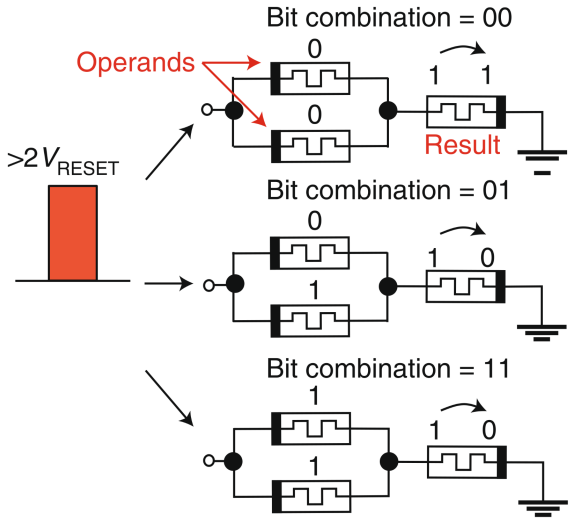



**Figure 2.4.:** Stateful CiM example: NOR operation using the MAGIC scheme [51].

non-stateful) CiM, in which operands are stored in the array but computation manifests as transient electrical signals at the periphery without modifying the stored states [53, 55, 148]. Both paradigms exploit the physical properties of memory devices and array organization, but they exhibit distinct trade-offs in terms of performance, endurance, energy, precision, and architectural integration.

### 2.3.1. Stateful CiM

Stateful CiM exploits the non-volatile resistance state of memristive devices as both storage and logic domains. Boolean variables are encoded directly in the device resistance (for example, LRS for logic '1' and HRS for logic '0'), and logic operations are realized by applying appropriate voltage pulses across devices inside the memory array. Logical operations are thus enabled through the interaction between the applied voltages and the state variables of the devices [146, 147].

Representative stateful CiM schemes include *MAGIC* and *IMPLY*, which realize Boolean functions such as NOR directly in resistive crossbar arrays [51, 149, 150]. In a typical MAGIC NOR gate, two input devices and one output device participate in a single operation: operand values are stored as resistance states in the input devices, a fixed initial state is written into the output device, and then a specific voltage bias pattern is applied to realize the NOR function. After the operation, the output device's resistance encodes the result, which can be immediately reused as input to subsequent gates. Figure 2.4 shows the NOR operation using the MAGIC scheme. Related 'stateful' logic concepts based on material implication and other primitives have also been demonstrated with memristive technologies [150, 151].

The main advantages of stateful CiM include:

**Tight memory–logic co-location:** The same physical devices act as both storage and logic elements, and all operands and results remain within the memory array, avoiding transfers to an external Arithmetic Logic Unit (ALU) [51, 151].

**Cascadability:** Since outputs are written as resistance states, they can directly drive follow-on gates, enabling more complex logic functions and pipelines to be realized entirely inside the array [51, 150].

**Parallelism:** Many gates can be activated simultaneously across rows or columns by broadcast control signals, exploiting the regularity and parallel nature of crossbar arrays [148, 150].

However, stateful CiM also exhibits significant drawbacks:

**Endurance and write stress:** Every logic evaluation involves writing at least one device, and multi-step functions can require many such writes per operation. This stresses devices with limited write endurance (e.g., ReRAM and PCM) [55, 148].

**Energy and latency per operation:** Logic is implemented via write-like pulse sequences, which are more energy- and time-consuming than simple read operations [55, 151].

**Variability and accuracy:** Reliable operation demands well-separated resistance levels, controlled write variability, and limited temporal drift, which can be challenged by filamentary switching and conductance fluctuations in nanoscale devices [55, 147].

Consequently, stateful CiM is attractive for tightly coupled logic-in-memory functions where minimizing data movement is the primary objective, such as simple database queries or bit-wise processing near stored data [28, 150]. Its reliance on frequent state updates, however, can be a limiting factor for very deep or frequently evaluated logic pipelines unless device endurance and write variability are carefully engineered [55, 148].

### 2.3.2. Stateless CiM

Stateless CiM decouples operand storage from result representation. Operands are stored as resistance values (or, in charge-based memories, as stored charge), but the result of a computation is produced as a transient electrical signal (typically a voltage or current sensed at the Bit Lines (BLs) or Word Lines (WLs)) rather than as an updated device state. The stored data remain unchanged during the operation [53, 55].

In a typical stateless CiM primitive, a group of cells representing input bits is biased with a specific WL/BL pattern; the resulting BL voltage or current encodes the outcome of a logic or arithmetic function, such as NAND, NOR, population count, or a partial sum in a MAC operation. Dedicated SAs or Analog-to-Digital Converters (ADCs) at the periphery detect this signal and convert it to digital form. DRAM-based CiM exploiting charge sharing among multiple cells on a BL, and memristive crossbar-based analog Matrix–Vector Multiplication (MVM) used for deep learning inference and compressed sensing, are prominent examples of such stateless behavior [46, 47, 52, 53, 55].

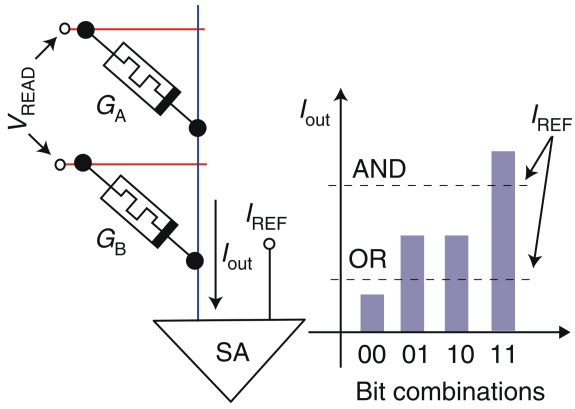



**Figure 2.5.:** Stateless CiM example: AND/OR operation using the scouting logic scheme [152].

Within this stateless class, *scouting logic* plays a central role as a memristor-based logic design for resistive computing [152]. Rather than rewriting device states to encode intermediate results, scouting logic carefully engineers the sensing and biasing scheme so that bulk bit-wise Boolean operations (e.g., AND, OR, XOR) are realized through the transient response of the crossbar array while preserving the stored conductance values. This allows many bits to be processed in parallel with read-like stress, significantly reducing write-induced wear and improving throughput for large in-memory logic operations [52, 152]. Figure 2.5 shows the AND/OR operation using the scouting logic scheme. The operation procedure is detailed in Section 2.5 when reliability challenges are discussed in depth.

Key advantages of stateless CiM include:

**No write wear during computation:** Since cell states are not modified, operations impose read-like stress on the devices, relaxing endurance requirements and enabling a large number of operations over the array lifetime [55, 148, 152].

**Energy efficiency for read-centric workloads:** Read-based or charge-sharing operations typically consume less energy than repeated programming pulses, particularly when many computations reuse the same stored data (e.g., inference-only NNs) [28, 46, 47, 53].

**Simplified reliability management:** Variability primarily affects read margins and analogue precision, which can be mitigated with established sensing, calibration, and error-correction techniques [53, 55].

These properties make stateless CiM particularly suitable for:

**High-throughput linear and bit-wise operations:** Inner products, MVMs, and bitwise logic used in signal processing, optimization, and machine learning workloads can be accelerated by analog computation in crossbar arrays, followed by digitization at the periphery [53, 55, 148, 152].

**Read-mostly workloads:** Applications where weights or lookup tables remain largely constant (e.g., deep NN inference or compressed sensing with fixed measurement matrices) benefit from repeated computations without rewriting the array [28, 53].

**Integration with digital accelerators:** Because outputs are delivered as voltages or currents that are digitized, stateless CiM blocks integrate naturally with conventional digital processing pipelines [28, 55].

The main challenges for stateless CiM arise from:

**Peripheral overhead:** ADCs or SAs required for accurate digitization consume area and power, and multiplexing them across many columns increases latency [28, 53].

**Analog non-idealities:** Wire resistance, device current–voltage nonlinearity, and conductance noise degrade computational precision and limit the practical size and bit-resolution of crossbar-based operations [53, 55].

**Multi-stage logic mapping:** While basic bit-wise and arithmetic primitives are natural, implementing complex multi-stage logic often requires sequencing operations and moving intermediate results between the array and peripheral digital units [28, 150, 152].

In summary, stateful CiM uses memory devices as both storage and logic, updating resistance states to encode results and enabling in-array logic fabrics [51, 151], while stateless CiM, including scouting logic and other read-based schemes, keeps device states fixed and computes via transient electrical responses, better matching high-throughput analogue MVM and read-centric workloads [53, 55, 152]. Together, these paradigms span a rich design space for in-memory computing architectures targeting diverse applications, from logic-in-memory to deep learning and scientific computing [28, 148]. However, in this dissertation, we specifically focus on scouting-logic-based CiM due to the benefits mentioned and, importantly, its reliability advantages.

## 2.4. NVM-CAM

In this section, we briefly introduce NVM-CAMs, including their basic cell organization and sensing principles, as a concrete example of resistive CAM. A detailed analysis of NVM-CiM-based similarity computation, covering a broad range of similarity kernels and providing an in-depth comparison of different similarity-computation flavors based on NVM devices in terms of performance, energy, and reliability, is deferred to Chapter 5.

Figure 2.6 presents a simplified STT-MRAM-based CAM structure consisting of 2T2R MTJ cells. Unlike conventional SRAM-based CAMs, where Match Lines (MLs) are discharged only in the event of a mismatch, each 2T2R cell inherently provides a continuous discharge path via one of its two resistive elements regardless of the match condition. In other words, each 2T2R cell effectively functions as an XNOR gate. When the query bit ($Q_j$) matches the stored bit ($D_{ij}$), the activated pull-down path includes an HRS, resulting in slower discharge. Conversely, a mismatch activates an LRS, leading to a faster discharge.

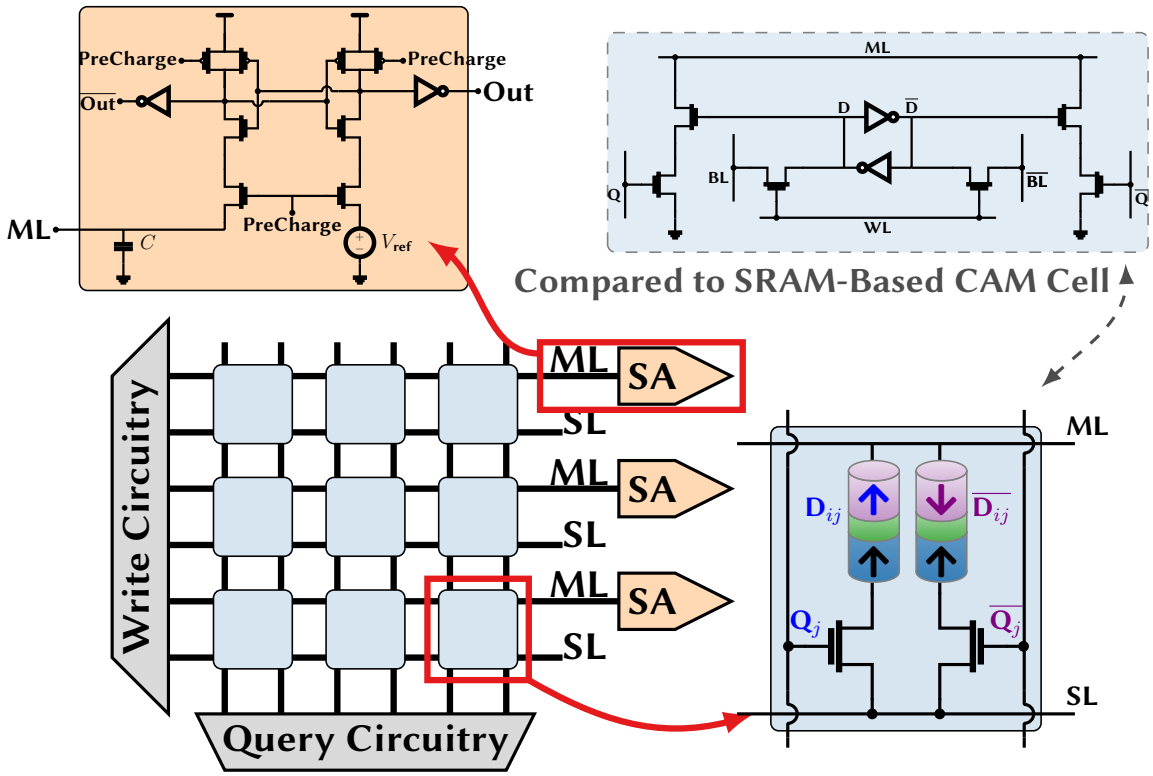



**Figure 2.6.:** Simplified STT-MRAM-based CAM structure featuring 2T2R MTJ cells compared to a conventional 10T SRAM-based CAM cell. The SA circuit is also shown.

### 2.4.1. Write Operation

Writing to a row is performed in two cycles. In the first cycle, all bits with a logical value of '1' are programmed. The query circuitry identifies these bits and enables the corresponding access transistors. The write circuitry then applies an appropriate write pulse across the ML and Source Line (SL) to program the targeted MTJs into the HRS. In the second cycle, the query values are inverted, and a complementary write pulse is applied to program the remaining bits into the LRS. This two-step approach ensures that all cells are correctly written according to their intended logic values.

### 2.4.2. Search Operation

The search operation also proceeds in two cycles. In the first cycle, all query outputs are set to '0', keeping the cell transistors disabled. Simultaneously, all MLs are precharged to $V_{dd}$ via dedicated precharge transistors. In the second cycle, the MLs are disconnected from $V_{dd}$, and the SLs are grounded. The query circuitry then drives the appropriate search lines to activate the relevant cell transistors. This enables the pull-down paths, causing the MLs to discharge. The rate of discharge for each ML depends on the number of mismatched bits in the corresponding row. At a designated sampling time ($t_s$), each SA compares the

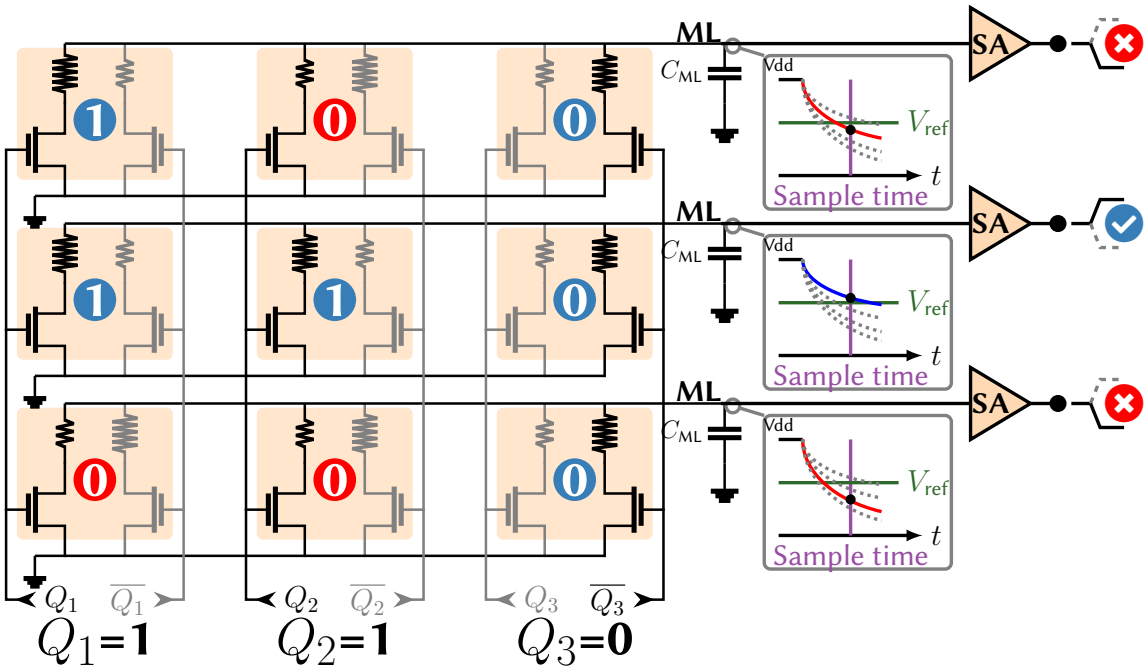



**Figure 2.7.:** Simplified search operation of an STT-MRAM-based CAM. Large (small) resistors represent MTJs in the HRS (LRS) state. Binary values indicate stored cell contents, and disabled paths are shown in gray.

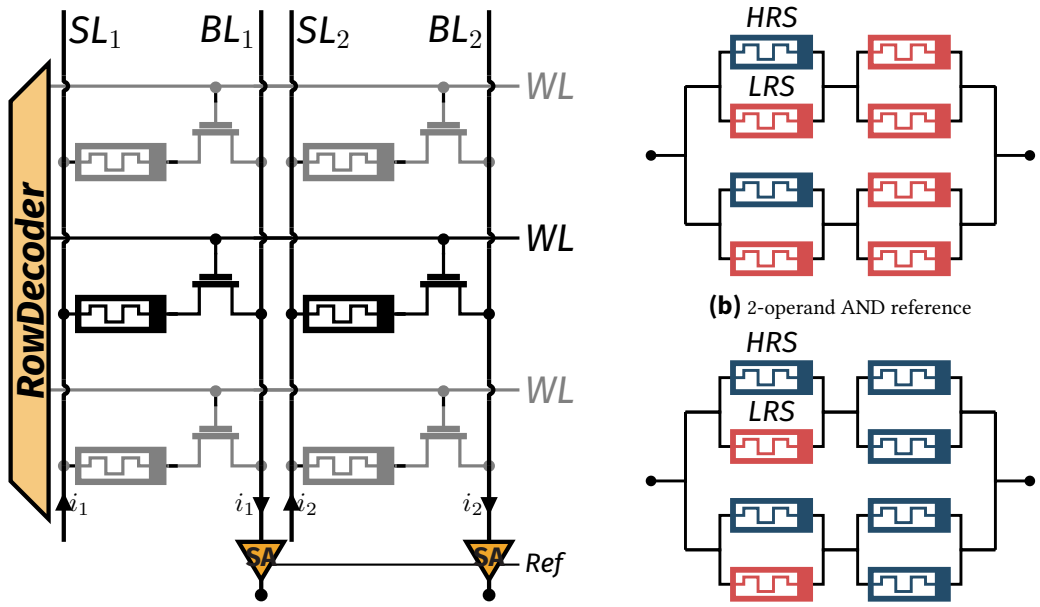
ML voltage against a global, fixed, reference/threshold voltage ($V_{ref}$), producing a binary match/mismatch output for the corresponding row.

Figure 2.7 illustrates this process. Large and small resistors represent MTJs in the HRS and LRS states, respectively. Binary values within circles indicate stored bits. Inactive paths are shaded gray. The effective resistance of the active pull-down paths determines the ML discharge rate. The SA samples the ML voltage at $t_s$ to determine whether the row matches the query.

The values of $t_s$ and $V_{ref}$ are chosen during circuit design by analyzing the voltage trajectories of matching and mismatching cases. The goal is to select $t_s$ such that the voltage difference between the two cases is maximized, and to set $V_{ref}$ as the midpoint between them. These parameters depend on factors such as the technology node, CAM array size, and design constraints.

## 2.5. NVM-CiM Reliability Challenges

A scouting-logic-based NVM-CiM architecture typically consists of a dense NVM crossbar array and peripheral CMOS circuits. The crossbar forms a two-dimensional grid of memory cells, commonly implemented using a 1T1R structure and accessed through shared WLs, BLs, and SLs, as shown in Figure 2.8a. The WL selects a row, the SL provides the read/write current, and the BL collects the resulting current. Each cell stores data by being




**(a)** Conventional NVM crossbar array

**(b)** 2-operand AND reference

**(c)** 2-operand OR reference

**Figure 2.8.:** Crossbar array of a conventional NVM memory. Trim reference generation [153] schemes for 2-operand AND and OR operations.

programmed to either a HRS or LRS. During reads, the BL current is compared with a reference current at the SA to determine the stored value.

Each bit-vector is stored in separate but column-wise aligned rows (Figure 2.9a–Figure 2.9b). These rows are activated simultaneously, causing the resulting currents to accumulate on the BLs (Figure 2.9c). During sensing, the accumulated BL current is compared with a reference current, and the output is set to '0' if the current is below the reference and to '1' otherwise. The result is then written back to memory (Figure 2.9d). Reference currents are typically generated using trim circuitry [153], which can track temperature and process variations (Figure 2.8b–2.8c). All crossbars within the same region share a common reference generation block, ensuring consistent sensing while reducing area overhead.

Despite these mechanisms, NVM-CiM relies on analog sensing of resistance states, introducing three main sources of error. First, memory cells deviate from their nominal resistance due to process variation and read/write effects, typically modeled as normal distributions [154]. As shown in Figure 2.10a–2.10b, this variation affects the parallel resistance of activated rows, causing distribution overlap and potential *decision failures* at the SA outputs. Second, imperfections in the SA and reference generation circuitry can introduce comparison inaccuracies, further reducing output distinction (Figure 2.10c). Third, increasing the number of activated rows (operands) produces more output distributions while reducing their average separation, further increasing error probability (Figure 2.10a–2.10b).

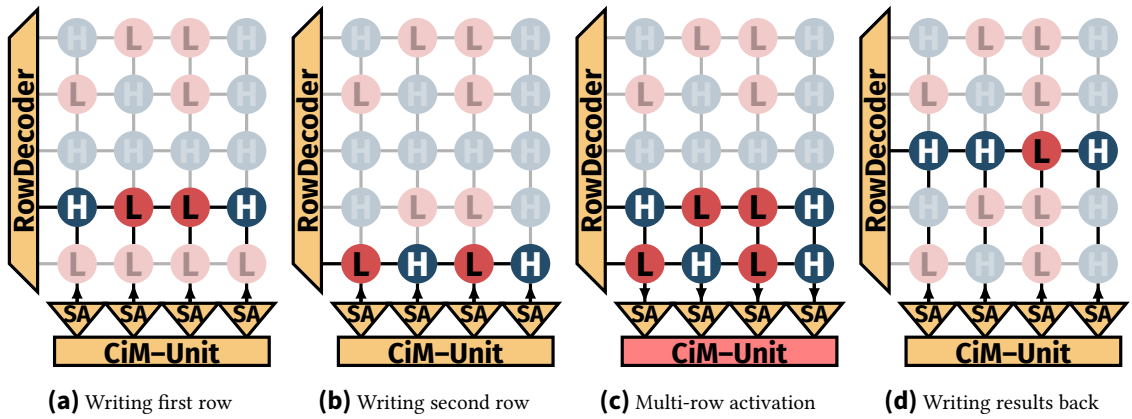



**Figure 2.9.:** Steps to perform a single scouting-logic-based Boolean operation.

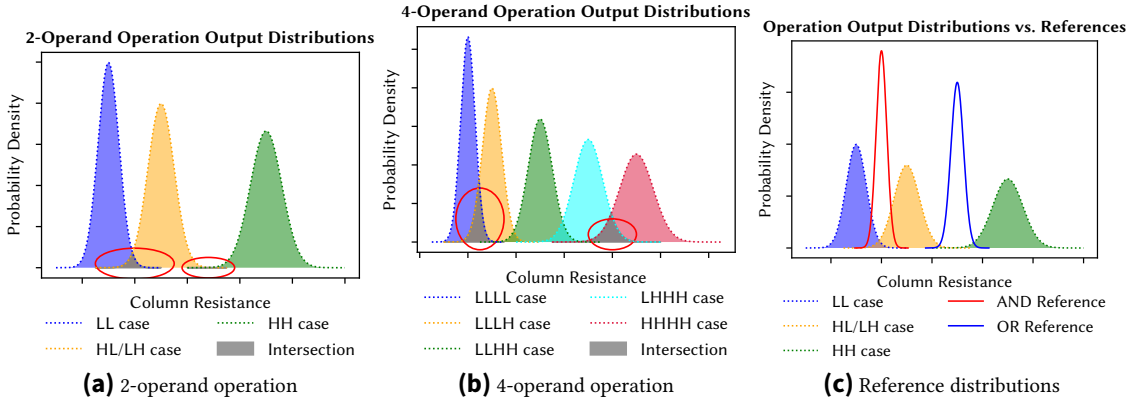



**Figure 2.10.:** Resistance variation distributions for 2- and 4-operand operations, and reference circuitry.

In general, the first source of error arises from the distributions of HRS and LRS, specifically the difference between their mean values, as well as the standard deviation of both distributions, which are directly linked to the NVM technology node. The second stems from the sensing techniques and the specific design approaches employed. Finally, the third depends on the CiM operation type and the number of operands, which are dictated by the application requirements.

Similar to scouting logic, the MAC operation is also susceptible to decision failure; however, due to the ADC requirement and its implementation, the mechanism of decision failure differs from that of scouting logic. As shown in Figure 2.11, the distribution of the BL current introduces a probability that the digital ADC output deviates from the expected value. Unlike scouting logic, decision failures in the case of the MAC are not primarily sensitive to the distance between two distinct distributions, but rather to the variation of the BL current distribution and the quantization level of the ADC ($Q_{\text{ADC}}$). The higher the BL current variation (i.e., the larger the standard deviation), the higher the error probability. Figure 2.11 also shows that the larger the quantization level of the ADC ($Q_{\text{large}}$ instead of $Q_{\text{small}}$), the lower the error probability. In the case of MAC operation, the error probability is data-dependent, since each combination of $[V_1 \dots V_N]$ and $[G_1 \dots G_N]^T$ leads

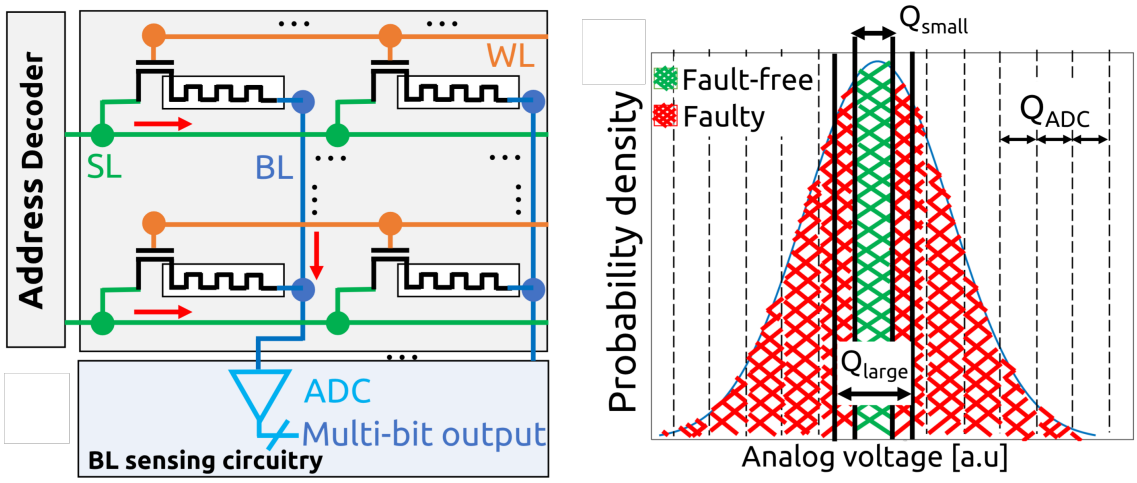



**Figure 2.11.:** Performing the MAC operation in the crossbar organization of NVM. Decision failure in the case of MAC operation utilizing ADC; the $Q_{\text{ADC}}$ shows the quantization level of the ADC, the probability associated with the green region is for the fault-free scenarios, while the red regions show the probability of decision failure; also, enlarging the $Q_{\text{ADC}}$ ($Q_{\text{large}}$ instead of $Q_{\text{small}}$), i.e., fewer bits for ADC quantization, reduces the probability of decision failure.

to a different standard deviation. Smaller MAC results are associated with less variable current distributions, which are substantially less prone to decision failure.

## 2.6. NVM-CAM Reliability Challenges

Despite the advantages discussed in Section 2.4, NVM-CAMs encounter significant reliability challenges due to device-level variability and manufacturing non-idealities. These factors cause deviations in bit-cell resistance from nominal values. In practice, the resistances exhibit statistical variation, often modeled by normal distributions, rather than remaining constant. This variability can introduce sensing errors and consequently increase the Search Error Rate (SeER) [79, 93, 123, 133].

Figure 2.12 extends the example of Figure 2.7 by incorporating the effects of process variation. In this scenario, device-level variations slightly decrease the HRS values in a match case. Simultaneously, both HRS and LRS values in a mismatch case are slightly increased. These shifts impact the effective pull-down resistances on the ML: the match case becomes marginally weaker (lower resistance), while the mismatch case becomes marginally stronger (higher resistance). Consequently, the ML in the match case discharges faster than nominal, and the mismatch case discharges more slowly.

Taken together, the background on memory technologies, in-memory computing paradigms, and reliability challenges provided in this chapter forms the foundation for the remainder of this dissertation. In the next chapters, we build on these concepts to design, benchmark, and protect NVM-based CiM and CAM architectures.

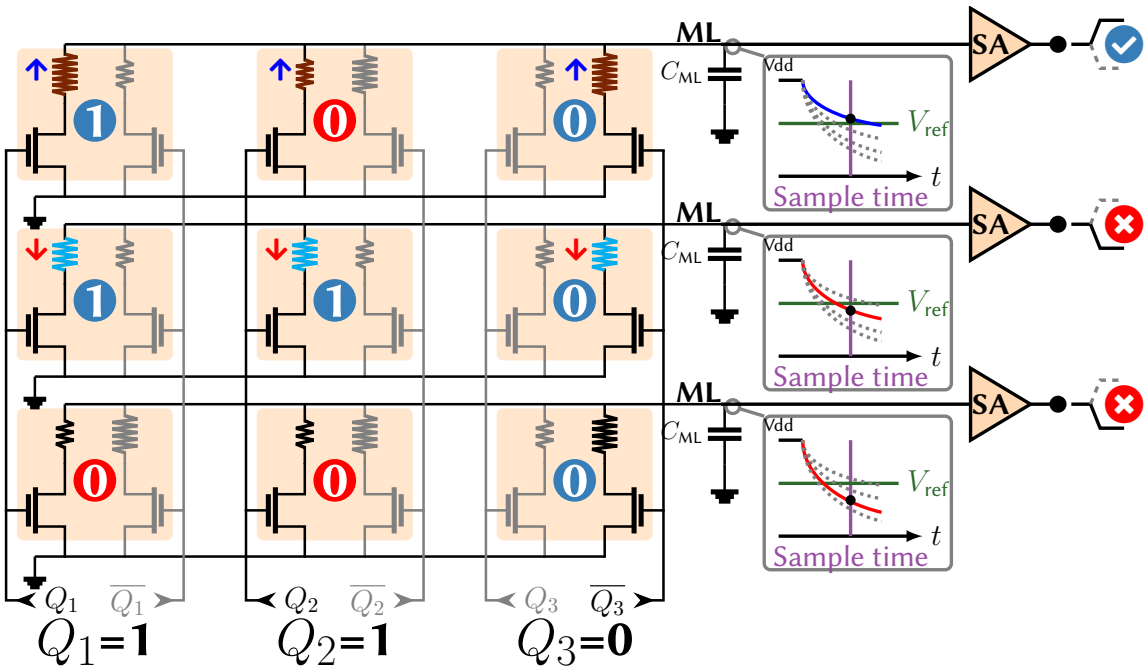



**Figure 2.12.:** Incorrect match detection in NVM-CAM caused by process variation.

# Part II.

# Contributions

# 3. Benchmarking NVM-CiM Workloads

> **Literature**
> This chapter is based on the following authored publication:
> "Sisyphus: Cross-Layer Efficiency Across NVM Technologies in Compute-in-Memory Architectures"

As discussed in Chapter 2, NVM-CiM architectures suffer from significant reliability challenges. Consequently, conventional schemes must be revisited and extended to provide the necessary protection [67, 68, 69]. At the same time, existing tools are limited in their ability to enable quantitative comparison of different memory technologies and architectural design choices under realistic workloads. To address this need, several CiM simulators have been proposed; they are qualitatively compared in Table 3.1 and further detailed in Section 3.2. However, to the best of our knowledge, and as demonstrated in Table 3.1, **no existing CiM simulator** encompasses all the critical features necessary for simulating a modern system in real-world environments. These essential features include (among others) full-system simulation support, extensive configurability of both technology and architecture, and, most importantly, robust reliability assessment.

We therefore propose *Sisyphus*, which uniquely fulfills these requirements and serves as a state-of-the-art solution for assessing cross-layer efficiency. Sisyphus allows researchers to thoroughly evaluate new protection or detection schemes—from device-level through algorithmic- and application-level solutions—and provides reliable insights for these emerging technologies. It is a novel, holistic framework that incorporates technology data, microarchitecture, latencies, power consumption, and Statistical Fault Injection (SFI) [155] to evaluate all aspects of modern System-on-Chip (SoC) architectures that use CiM accelerators. Sisyphus is built on top of gem5 [156, 157, 158], the state-of-the-art full-system microarchitectural simulator used by both academia and industry. By incorporating detailed technology- and circuit-level information of CiM modules (in terms of technology-relevant speed and power metrics as well as fault behaviors) and abstracting them at the microarchitecture level, we can accurately assess the performance, power, and reliability of various applications while running on a CiM-enabled architecture. Additionally, Sisyphus provides flexibility and fast design-space exploration across abstraction layers, from the NVM technologies up to the hybrid (computational and storage) memory organization, enabling comprehensive evaluation. The major contributions of this chapter are:

1. We extend the gem5 simulator to support highly configurable CiM architectures based on STT-MRAM, ReRAM, and PCM, the prevailing NVM technologies. This

**Table 3.1.:** A comparison of well-known CiM simulators.

| Simulator | Technology | Full-System Support | Architecture | Detailed Circuit Model | Reliability Analysis |
|---|---|---|---|---|---|
| CIM-SIM [161] | NVM | ✖ | Boolean, NN | ✖ | ✖ |
| MNSIM [162] | ReRAM | ✖ | NN | ✔ | ✖ |
| NeuroSim [163] | ReRAM | ✖ | NN | ✔ | ✖ |
| PIMulator [164] | FPGA, SoC | ✔ | Complex FPGA | ✖ | ✖ |
| MultiPIM [165] | CMOS | ✔ | Complex | ✖ | ✖ |
| SIM2PIM [166] | CMOS | ✔ | Complex | ✖ | ✖ |
| PIMSim [167] | CMOS, NVM | ✔ | Complex | Rely on other | ✖ |
| **Sisyphus** | NVM | ✔ | Flexible | ✔ | ✔ |

extension allows accurate performance, power, and reliability evaluations, enabling broader design space exploration.

2. We develop and provide a comprehensive Application Programming Interface (API) for the user application code to call and use different CiM operations.
3. Sisyphus implements SFI at the microarchitecture level for all hardware structures of the CPU and the CiM module operations. To the best of our knowledge, Sisyphus is the first SFI framework for CiM-enabled architectures.
4. Sisyphus reports accurate power estimation for CiM architectures by providing workload-dependent on-chip as well as memory and CiM power consumption by extending the widely used McPAT framework [159] along with accurate NVM models from NVSim [160] with CiM extensions.

As a showcase, this study reveals that the proposed CiM-assisted infrastructure significantly enhances performance, offering over 7× latency improvement and 19% power savings compared to CiM-free architectures when executing CiM applications. Our analysis highlights a complex energy–performance interaction, with CiM consistently surpassing CPU-only configurations in energy–delay metrics. However, the reliability analysis underscores divergent trends among NVM technologies: ReRAM emerges as the most robust option, while STT-MRAM demonstrates lower resilience. Combining these dimensions using a novel Efficiency-Resilience Coefficient (ERC) metric shows that application-specific variability drives CiM's performance potential and underscores the need for adaptable, cross-layer system modeling. This holistic approach highlights CiM's promise in high-end systems while emphasizing the importance of carefully tailoring memory technology and workload pairings for optimal trade-offs.

## 3.1. Reliability Assessment of CPU

While this chapter focuses on NVM-CiM reliability, the reliability of the host CPU must also be quantified. Modern CPUs are vulnerable to transient soft errors caused by effects such as radiation or voltage fluctuations, which may affect caches, register files, and pipeline state. Depending on timing and workload behavior, these faults may be masked or propagate to program-visible failures, including crashes and Silent Data Corruptions (SDCs). To establish a realistic reliability reference, we perform statistical microarchitectural fault injection, enabling full-system observation of fault propagation across hardware and software layers. This provides a CPU-only reliability baseline for comparison with CiM-based execution.

### 3.1.1. Early Reliability Assessment

Early reliability assessment typically occurs using models that are developed prior to the creation of silicon prototypes. These models can be categorized into three main levels: architecture, microarchitecture, and Register-Transfer Level (RTL). These levels vary in terms of detail, with the most abstract available early in the design process and the most detailed (RTL) emerging later on [168, 169]. Architecture-level models often lack most, if not all, hardware-specific details, providing a functional emulation at the software level [170]. In contrast, microarchitecture-level models (e.g., based on gem5) encompass functional and timing-accurate representations of the microarchitecture and offer accuracy at the cycle level. Additionally, microarchitecture-level models accurately depict various memory elements within the system, such as SRAMs, registers, and non-logic-related flops and latches, such as state machines. RTL models provide a comprehensive description of the implemented hardware, encompassing logic components and SRAM elements. Simulation time required for each abstraction layer correlates with the level of detail, with each increase in detail adding approximately two orders of magnitude to the simulation time.

### 3.1.2. Microarchitecture-Level Fault Injection

Typically, designers use either SFI [155] or analytical methods like the Architecturally Correct Execution (ACE) analysis [171] to gain insights into the resilience of programs against transient faults[1]. For transient faults, both methods aim to measure the Architectural Vulnerability Factor (AVF) of hardware structures, but each has its advantages and disadvantages. SFI is very accurate, but slow since it requires multiple runs to reach high confidence, whereas ACE analysis is fast, but has a high development effort and may overestimate AVF (up to 3× overestimation [172]). SFI is a reliability estimation technique

[1] ACE applies to transient faults only, while fault injection works for both transient and permanent faults since the entire execution of the program is simulated in the presence of faults.

that can provide full-system AVF estimation by directly accessing the program output produced with the injected faults. SFI is a widely adopted method for reliability assessment. It offers flexibility in terms of accuracy, depending on the size of the statistical sample, while providing failure samples generated through simulation. However, it comes with the drawback of requiring multiple simulations, which can be time-consuming and, depending on the level of model detail, may be considered impractical.

## 3.2. Related Work

Multiple research efforts aim to enhance the reliability of NVM-CiM. The work in [173] focused on hardware-level decision failures in scouting logic, proposing signal fine-tuning to mitigate these failures. Similarly, [174] introduced voltage tuning to address decision failures. However, these studies remained limited to technology and circuit levels, failing to connect to architecture and application levels. Other works [175, 176, 177, 178] targeted MAC-based NVM-CiM, prioritizing IR drop[2] reliability. Despite their comprehensive analysis across technology, circuit, and application levels, they overlooked architectural analysis. George *et al.* [179] argued that a microarchitecture-level SFI approach can provide accurate vulnerability estimations and proposed an SFI framework on top of the PTLSim [180]. Yalcin *et al.* [181] proposed FIMSIM, an SFI framework for microarchitectural simulators. While that framework provided fault injection support for various structures, it mainly targeted the M5 simulator, which has been superseded by gem5 and was evaluated only with simple in-order cores of the Alpha Instruction Set Architecture (ISA). Kaliorakis *et al.* [182] proposed GeFIN, an SFI framework built on top of gem5. GeFIN supports injections on different microprocessor structures, but not for any kind of accelerator, and targets a rather old version of gem5. Apart from the microarchitecture-level frameworks, which are very limited and target only the CPU structures, prior research [183] has shown that Intermediate Representation (IR)-level and assembly-level injections deliver very similar results. Venkatagir *et al.* [184] introduced gem5-Approxilyzer, which operates at the architectural level and precisely evaluates how an instruction error affects the final output. However, these approaches at the software or ISA layer may not fully capture critical fault manifestation and propagation aspects, potentially leading to misleading reliability findings, as was recently demonstrated in [170].

Additionally, it is worthwhile to consider other CiM simulators to gain a better understanding of what Sisyphus offers. Among well-known CiM simulators: CIM-SIM [161] is a functional behavior simulator for CiM architectures, written in SystemC. Although it is technology-agnostic, it is limited to its proposed "Nano-Architecture". MNSIM [162] focuses on simulating memristor-based neuromorphic acceleration. It operates at the memory-array level and aims to accelerate SPICE-level simulations. NeuroSim [163] targets NN acceleration tasks using CiM, based on 40 nm ReRAM technology. However, it

[2] IR drop (or ohmic drop) is the voltage reduction ($V=IR$) that occurs as current flows through the resistance of Power Distribution Networks (PDNs).

lacks configurability, and evaluations are restricted to the provided design. PIMulator [164] uses Field-Programmable Gate Array (FPGA) to emulate PiM architectures by integrating PiM Intellectual Propertys (IPs) between the memory controller and the main memory path. MultiPIM [165] is a configurable PiM simulator that can be integrated with full-system simulators, allowing for PiM timing simulations using trace files. Sim2PIM [166] is a high-level, PiM-agnostic simulator that enables fast PiM design profiling. PIMSim [167] is a highly reconfigurable full-system PiM simulator that relies on gem5's functionality, offering trade-offs between simulation accuracy and speed. None of the listed simulators offers reliability assessment, and Sisyphus tries to fill this gap.

## 3.3. CiM System Architecture

The majority of literature on NVM-CiM architectures [52, 185, 186, 187] focuses on very specific tasks, such as accelerating NNs or facilitating basic Boolean operations. However, this specialization limits the flexibility of diverse application implementations. Further, with these custom architectures, it is difficult to effectively model and analyze circuit-level factors (such as power consumption, latency, and error rates) in high-level simulations. To close this gap, we develop a general-purpose architecture to address these concerns and assess the reliability impact across different applications. In the following, we discuss the CiM-enabled architecture of Sisyphus and the proposed memory organization consisting of both storage and computational memories. Additionally, we discuss how the proposed architecture can support the execution of various CiM instructions.

### 3.3.1. Memory Organization

Figure 3.1 presents a high-level representation of a conventional full-system computer connected to the main memory module, highlighting our modifications. We assume that the main memory module is a well-established Load-Reduced Dual Inline Memory Module (LRDIMM), which enables intermediate buffering and separates the direct signal connection between the CPU and memory chips via its controller. To enable CiM, we extend this controller to filter out CiM commands or data (similar to regular data sent from the CPU to the main memory, but distinguishable by their addresses) and forward them to the CiM controller.

The CiM controller, consisting of a basic state machine, fetches and directs the hardwired signals embedded within the CiM command to dedicated components in the CiM-capable chip while preserving the timing order. The CiM-capable chip is similar to a conventional memory chip, but with modifications: the row decoders can activate multiple rows simultaneously, the SAs are enhanced to support subarray level analog operations, and additional CMOS-based CiM logic circuitry is added to the output of the SAs buffers. Figure 3.2 illustrates the internal structure of the CiM-capable chip along with these modifications. Additionally, we have integrated a Direct Memory Access (DMA) module into the CiM controller to facilitate data movement between the memory chips and the CiM-capable

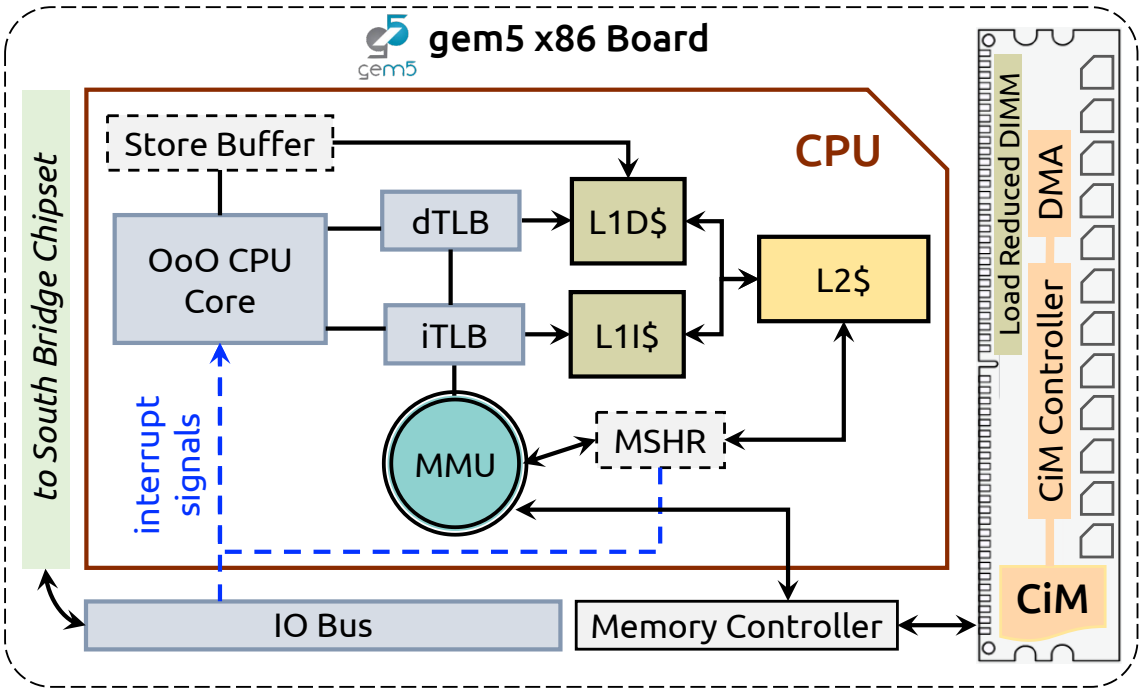



**Figure 3.1.:** High-level representation of a conventional full-system computer. The LRDIMM-based main memory controller is enhanced with a CiM controller and internal DMA, enabling efficient bulk data movement between CiM-capable logic and memory chips.

chip, eliminating the need for CPU engagement. The motivation behind this decision is explained in Section 3.3.3.

### 3.3.2. CiM Operations

Data-intensive workloads are the most suitable applications for taking advantage of CiM capabilities. These tasks can be parallelized through simple SIMD operations like AND, OR, XOR, MAC, etc. Boolean CiM operations are implemented with the scouting concept using a comparator that is already adjusted based on the CiM operation. The MAC operation utilizes a specialized ADC. However, relying solely on these operations is not enough to effectively harness CiM capabilities. The continuous data exchange between the CPU and CiM incurs more time and energy than calculating these operations internally within the CPU. *Embarrassingly Parallel* [188] workloads allow most of their execution to be offloaded to CiM units. The CPU can then use the prepared results to carry out the final post-processing tasks. In Sisyphus, activation of two, four, or eight rows is supported for this set of operations. Although in some NVM technologies, it is possible to activate many rows, for most applications this is satisfactory.

To properly execute the primary portion of these workloads in the CiM, it is crucial to have a few specific operations beyond the basic operations described above. These operations include a COPY operation that can rotate data, a bitwise NOT operation, and a byte-wise comparison by `0x00`. Byte-wise comparison by `0x00` involves performing an OR operation

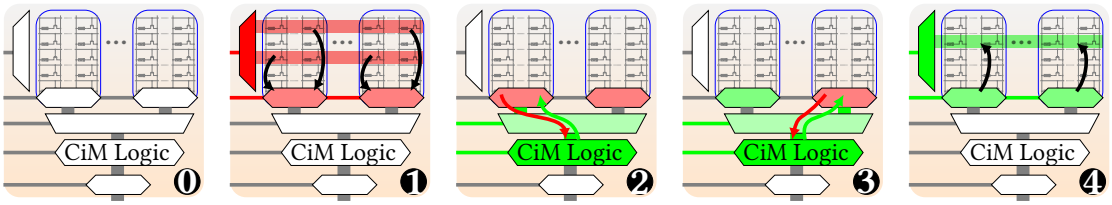



**Figure 3.2.:** Execution steps of a bitwise NAND operation in our proposed CiM units.

**Table 3.2.:** Essential operations supported by the CiM extension.

| Operation Type | Operation | Source | Destination | Description |
|---|---|---|---|---|
| In Memory Array | `AND, OR, XOR` | A List of Rows | A Single Row (Default: `SA` output) | Perform bitwise operation |
| In CMOS Periphery | `COPY` | A Single Row (Default: `SA` output) | A Single Row | Copies row content Optional: bitwise rotation |
| | `NOT_COND` | | | Performs bitwise NOT Or vector mask operation. |
| In CiM Controller Or DMA-Related | `copy_to_cim` | A pointer to a vector | CiM row number | Copies vector to/from a CiM row If DMA: Main Memory ↔ CiM |
| | `copy_to_cpu` | CiM row number | A pointer to a vector | |

on all 8 bits in a byte, which will result in either `0xff` or `0x00` in the corresponding output byte. The last operation is necessary to enable the implementation of basic `if` statements and is almost equivalent to the `PCMPEQ` command in the MMX extension for Intel architectures [189]. Table 3.2 summarizes these operations. Furthermore, to demonstrate the extensibility of the CiM framework, the binary MAC operation [50] has been added to enable MVM, a dominant operation in NN applications. The format of CiM instructions contains information regarding operation type and selected row indexes. It also includes various bit-fields within the instruction structure, allowing the programmer to have fine-grained control over the operations. Specifically, the programmer can set these bits to specify on which banks, subarrays, or the corresponding bytes in the subarrays a particular operation should be performed.

### 3.3.3. Interaction with CiM Module from the Application-layer

The CiM controller can be either active or inactive. When it is inactive, the LRDIMM controller can perform read and write operations in the CiM region in the same way as any other memory region. However, when the CiM controller is active, the LRDIMM controller must wait until the CiM controller completes its tasks before accessing the corresponding regions. To enable communication between the application layer and the CiM controller, we use a fixed physical shadow address, which is an address with no specific physical location mapping to it. Instructions from the application layer are written to this address, and when the LRDIMM controller detects them, it forwards the content directly to the CiM controller. It is important to note that the memory mapping is non-cacheable and non-shareable. Non-cacheability is essential because the CPU process writes data directly

**Listing 3.1:** Bitwise NAND operation on two vectors in our CiM API.

```
//row2 <- ~(row1 & row3)
cimModule.AND({ 1, 3}); // row1 & row3
cimModule.NOT_COND(2, true, false); // bitwise NOT
```

**Listing 3.2:** An example of a ternary operator in C++ language.

```
for(size_t i=0; i< MAX_ROW_SIZE; i++)
   a[i] = (b[i]==0x12) ? c[i] : d[i];
```

into the main memory. Non-sharability is also crucial to ensure data consistency and prevent a process from acquiring this hardware resource while another process has not released it. To enforce these requirements, Sisyphus provides an Operating System (OS) driver and an API. Listing 3.1 shows a simple code snippet of this API for a bitwise NAND operation on two vectors, while Figure 3.2 illustrates the actual operations within the CiM units.

In the code snippet shown in Listing 3.1, we assume that some content is already stored in row #1 and row #3 within the CiM units. The AND operation is then executed simultaneously inside the subarrays by activating two lines (row #1 and #3). Subsequently, the NOT operation is performed on the intermediate results using the CiM logic, and finally, the outcomes are written back to row #2. To demonstrate how to use the `if` statement in the CiM API, assume the ternary operator in C++ programming language, as shown in Listing 3.2. To convert this sequential C++ code into CiM instructions, we assume that each vector has a maximum size that equals the CiM row size. For our simulations, we also assume the presence of 16 banks in the CiM-capable chip, where each bank comprises 64 subarrays with a column size of 8 bytes. Therefore, in this example, we set the `MAX_ROW_SIZE` to 16×64×8=8 KiB. In case the vectors are larger than this value, they can be divided into aligned sizes.

The example code in Listing 3.3 demonstrates the use of the CiM API. In this approach, the programmer copies each vector to a dedicated row, which allows operations on each byte to be executed in parallel, eliminating the need for a `for` loop. If our application has other types of loops, they will be unrolled during the CiM code generation, since our CiM Controller does not include loop control instructions. Similarly to using SIMD instructions in assembly language, it is the programmer's responsibility to convert sequential code into a parallel version in the CiM context as well.

CiM applications operate independently of CPU interaction during CiM operations and have a fixed number of instructions. Therefore, issuing CiM instructions individually from the application level to load files into the cache and offloading them to the CiM region is inefficient and not in line with the CiM concept. A more efficient approach is to store all the CiM instructions in the main memory, trigger the CiM controller to read and execute these instructions, and perform intra-main memory data loading if required.

**Listing 3.3:** Converting sequential ternary operators to parallel CiM instructions.

```
cimModule.copy_to_cim(0, (void *)&b[0]); // row0 <- b...
cimModule.copy_to_cim(1, (void *)&CONSTx12[0]); // row1 <- 0x12...
cimModule.copy_to_cim(2, (void *)&c[0]); // row2 <- c...
cimModule.copy_to_cim(3, (void *)&d[0]); // row3 <- d...
cimModule.copy_to_cim(4, (void *)&a[0]); // row4 <- a...

cimModule.XOR({0,1}) // (b[i]==0x12)
// row5 <- 0xff if result is 0, otherwise 0x00
cimModule.NOT_COND(5, false, true);
cimModule.NOT_COND(6, 5, true, false); // row6 <- ~row5
// row7 <- row2 & row5
cimModule.AND({2,5})
cimModule.COPY(7)
// row8 <- row3 & row6
cimModule.AND({3,6})
cimModule.COPY(8)
// a <- row7 | row8
cimModule.OR({7,8})
cimModule.COPY(4)
```

To achieve this, internal DMA must be integrated, and the internal memory bus must be shared between the DMA and the LRDIMM controller.

Integrating DMA into the CiM controller imposes significant challenges in the hardware, OS, and application layers. Data consistency is the most critical challenge. The CiM controller manipulates some memory locations, while the CPU cache memory may have a copy of them. This copy may become dirty, and the CPU may want to store data back in memory when the cache is full. As a result, the CPU may overwrite the CiM results, or vice versa. Since there is no interaction between the data cache and CiM during the CiM operation, and the CPU only loads the final results into the cache to perform post-processing tasks, handling *cache coherency* becomes much easier. Additionally, the CiM controller must access some memory pages that may cause segmentation faults without being caught from the OS perspective. Since the OS can only check page access privileges at the CPU level, it is assumed that the programmer has complete control over the system and is aware of these challenges. More specifically, these are challenges that can be resolved at the application layer. The cache can be flushed before performing CiM operations, and this can be done with the assistance of both the compiler and CiM API.

## 3.4. Experimental Setup

To assess the performance, energy efficiency, and reliability of the various technologies and design decisions examined in this chapter, we present the modification of the cycle-level gem5 simulator with proper extensions for CiM, SFI, and power models.

### 3.4.1. CiM Extensions for gem5

To integrate CiM into the gem5 framework, we make minor modifications to the memory controller and memory interface libraries while preserving the integrity of the remaining gem5 source code. CiM controller functionalities have been added to the memory controller library, and technology-specific parameters are now part of the memory interfaces. Furthermore, to conduct a comprehensive simulation infrastructure, we employ a cross-layer technology-to-application flow in our framework. Starting at the technology level, we utilize realistic, measurement-validated Verilog-A models for the NVM devices. In both the scouting logic and MAC operations, the output is generated using resistance sensing. Therefore, the resistance distribution in LRS and HRS for each technology is relevant information, which is abstracted from the technology level and passed to the circuit level. Circuit-level simulations are performed using SPICE tools and an industry-aligned Process Design Kit (PDK). The circuit-level analysis outputs are ❶ latency and energy of the CiM modules via single-run SPICE simulation and ❷ error probabilities via Monte Carlo SPICE simulation. These parameters will be abstracted and passed to memory array simulation and microarchitecture-level fault injection, respectively. Besides, for specific requirements, such as the encoder for the flash-type ADC, latency and energy are derived by synthesizing the behavioral Verilog code using industrially aligned standard cell libraries.

At the memory array level, the NVSim tool is used to determine latency and energy, with updates according to the energy and latency of CiM-related modules, including the address decoder and SA, based on circuit-level analysis results. The behavior of memory subarrays and CiM operations is modeled at the system level within gem5. At the microarchitecture level, what changes between the different NVM technologies is the timing, the Bit Error Rate (BER), and the power/energy consumption of each operation. Therefore, gem5 and McPAT interfaces are augmented with accurate results from lower-level simulations (updated CiM-aware version of NVSim), ensuring a detailed cross-layer analysis and abstraction for accuracy and technology awareness. Furthermore, a CiM API is provided at the application level to access CiM functionality. While “address memory mapping” relates to the OS level, gem5 offers APIs to enable it through gem5 config files.

### 3.4.2. Microarchitecture-Level SFI

Here, we describe our microarchitecture-level evaluation approach and explore the challenges of applying SFI to CiM systems in Section 3.4.3. SFI should ideally be based on a real system with physical injection capabilities in all microarchitectural structures. However,

such a system does not currently exist. Even if it did, making protection decisions after studying and implementing it would be too late. Alternatively, a very detailed low-level simulator (e.g., RTL, gate) that allows the same can be used. While low-level simulators may provide accurate fault effects, their simulation throughput is extremely low and unaffordable. Additionally, they cannot model long-running workloads with OS activity. RTL simulation is several orders of magnitude slower than cycle-level microarchitectural simulation [190, 191, 192]. To address these challenges, Sisyphus relies on microarchitecture-level simulation using the latest version of the gem5 simulator. This approach enables ❶ deterministic, ❷ end-to-end, ❸ cycle-level execution of ❹ real workloads. This combination is impossible at lower levels. When it comes to CPU reliability evaluation, we follow the methodology as described in [155] and [170].

Briefly, all fault injection campaigns in Sisyphus (be it for the CPU or CiM module) consist of injecting a series of faults (statistical samples) into simulations. For statistically significant results, we inject 2,000 faults for every fault injection campaign. We follow the widely adopted formulation of [155] for the statistical fault sampling calculations; our 2,000 fault samples correspond to a 2.88% error margin with a 99% confidence level. These simulations produce output results, which are parsed to estimate the vulnerability or other desired metrics. Depending on the component being studied, an appropriate statistical sample is chosen. A high-level Python module serves the role of campaign controller and manages fault injection campaigns. Each fault injection simulation requires a fault description file that is produced by the campaign controller. This file defines the location and type of fault to be injected (for the calculation of AVF, single-bit-flip faults are chosen). The simulations are run, with the controller supplying necessary inputs and storing the results.

Sisyphus can configure multiple workers to achieve 100% hardware resource utilization. Each instance corresponds to one injected fault and produces output files and logs. Simulation outputs are stored for post-processing, determining the effect of the injected fault. Results are parsed, and finally, the evaluation metric estimation is exposed.

### 3.4.3. SFI on the CiM Module

The challenge in implementing SFI on the CiM module stems from the different underlying fault model that causes the majority of CiM errors. While in CPUs, a major fault type is transient Single-Bit-Upsets (SBUs) caused by alpha-particle and neutron radiation; the main source of error in CiM is internal and originates from the non-idealities discussed in Section 2.5, mainly affecting the different row-operations that the CiM supports (like AND, OR, XOR, MAC). The BER for each operation not only depends on the number of operands of the operation, but is also data-dependent. The decision failure that causes these errors originates from the overlap of the resistive states due to the process variation on the NVM devices, impacting the distribution of the output current. To find the data-dependent probability of the decision failure (in our case, the BER), we perform a Monte Carlo simulation on the NVM subarray while considering the effect of process variation on the NVM devices. By obtaining the distribution of the output current, the probability

of decision failure can be obtained from the overlap region, as shown in Figure 2.10. This fact makes the traditional SFI approach used for SRAM arrays in CPUs, where errors are injected following a uniform distribution across all the bits of the array, inaccurate and, hence, inapplicable.

To solve this issue, we inject single-bit flips weighed by the normalized average BER of each operation type (here, operation type refers to the actual operation and the number of rows/operands being activated/processed). Assuming we inject $N$ total faults, the number of faults injected to operation type $OP_i$ is given by Equation 3.1, where $BER_i$ is the average bit-error-rate of $OP_i$ across the entire execution of the workload being evaluated. To compute these average BERs, a single fault-free execution is done using a custom mode of our gem5-based SFI infrastructure. Models that capture the operand count dependence and data dependence of CiM operation BERs have been built using data from the technology and circuit layers and are employed during this golden run. By weighing the number of faults injected into each operation by the normalized BER, we guarantee that our fault sample is representative of the workload-memory technology combination being evaluated.

$$\text{Faults}_i = \frac{\text{BER}_i}{\sum_j \text{BER}_j} \times N \tag{3.1}$$

### 3.4.4. Full-System Power/Energy Modeling

To model the entire system power, we split it into *on-chip* and *off-chip* parts, as shown in Figure 3.3. The on-chip power is consumed by the CPU and caches, while the off-chip power is consumed by the (storage) memory and the CiM module. McPAT [159], a widely used tool, computes the on-chip average power using statistics traces from gem5 for each executed workload. To compute the off-chip power of the memory, we use NVSim [160], originally built on top of CACTI [193] with NVM extensions. However, NVSim is designed only for storage memory (read and write accesses) and needs to be extended for the CiM module. There are two main differences between NVSim and our CiM extensions. First, in the CiM-extended version, the row decoder module activates more than one row at a time, leading to scaled power and energy consumption. Second, the power and latency of the SA module need updating based on the number of activated rows and the NVM technology. By accurately modeling these components at the circuit level (i.e., SPICE) and performing circuit simulations, the corresponding models of NVSim are updated for the entire CiM module. Similarly, for the off-chip power, the statistics trace is required to calculate the average power of the normal memory and CiM based on their respective number and type of accesses. These accesses are multiplied by the per-access power obtained through NVSim and its CiM extension. The total power of the application running on Sisyphus full system is the sum of the on-chip and off-chip power contributions.

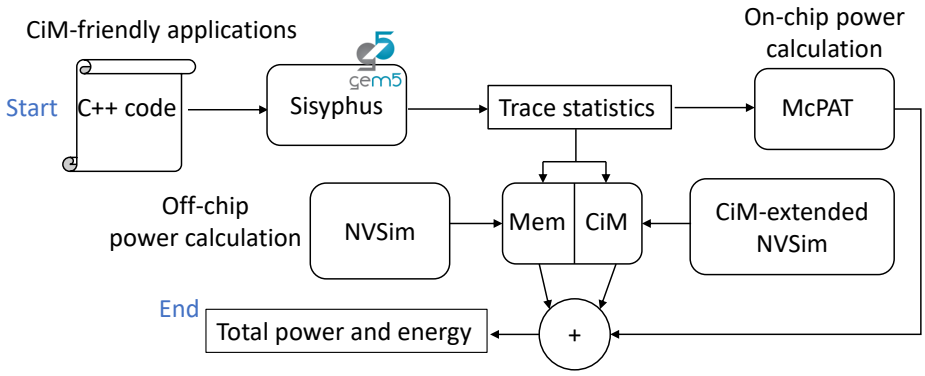



**Figure 3.3.:** Flow to calculate power and energy consumption.

### 3.4.5. Applications

We implement nine prevalent CiM-friendly applications:

1. **BitIndexing [39] (BIT):** Bitmap-formatted database queries are ORed with each other, and then the results are ANDed with another query. Finally, the number of activated bits is counted and reported as the final result.
2. **AES-ECB encryption algorithm [194] (AES):** There are four main computation phases: `AddRoundKey`, `SubBytes`, `ShiftROWs`, and `MixColumns`. During the different rounds of this algorithm, `AddRoundKey` and `MixColumns` are calculated inside the CiM module, and other phases are calculated inside the CPU.
3. **BLASTN algorithm [41] (BLS):** It is a well-known algorithm for searching short DNA sequences inside large DNA protein sequences.
4. **Morphological Image Processing [43] (MIP):** Various procedures can be applied to images during image processing. For this study, we implemented *Dilation* and *Erosion* algorithms with binary-coded input images with various filter sizes.
5. **Marching Squares [44] (MSQ):** It is an algorithm used for extracting contour lines from 2D images. In this chapter, we implement this algorithm for binary-coded input images.
6. **Shifted Hamming Distance [42] (SHD):** It is a well-known algorithm used for calculating edit distances in short DNA sequences. It accepts a minimum-distance number and checks two sequences against each other to see whether they satisfy this minimum-distance condition or the number of insertions, deletions, or mutations is larger than the minimum-distance number.
7. **BitWeaving [40] (BWV):** A technique presented to scan database queries in memory and return the results that satisfy the input conditions (e.g., $a \leq x < b$).
8. **Binary Matrix Multiplication [50] (MAC):** This application utilizes the MAC operation in our CiM framework to perform binary matrix multiplication. The size

**Table 3.3.:** Main Parameters of the Simulation Setup.

| | |
|---|---|
| NVM models | STT-MRAM [140]<br>ReRAM (JART VCM Read variability)<br>PCM [143, 144] |
| Subarray organization | 256 Rows, 64 columns |
| CPU type | OoO |
| Compiler & Optimization Flag | `g++ -O3` |
| ISA / Clock Frequency | x86-64 / 1 GHz |
| # of Load/Store/Issue/ROB Entries | 32 / 32 / 64 / 128 |
| # of Physical Int/FP/Vec Registers | 128 / 128 / 128 |
| L1 Inst/Data Cache Size | 32 KiB |
| L1 Tag/Data/Response Latency | 2/2/2 cycles |
| L2 Cache Size | 256 KiB |
| L2 Tag/Data/Response Latency | 20/20/20 cycles |
| System Bus Latency | 10 ns |
| Average Memory Controller Latency | 15 ns |
| Read Latency (STT-MRAM/ReRAM/PCM) | 1 cycle / 1 cycle / 1 cycle |
| Write Latency (STT-MRAM/ReRAM/PCM) | 4 cycles / 45 cycles / 40 cycles |

of the stationary matrix was kept fixed at 8×65536, and we used different matrix sizes for the non-stationary inputs.

9. **NN Inference [195] (MNIST):** utilizes a 4-layer NN with binarized weights and inputs (values of $-1, 1$), employing the $\text{sign}(x)$ activation function to perform inference on the MNIST[196] test dataset.

For every implemented application, we verify the CiM-enabled results by comparing them with the results from the CPU-only execution. This comparison confirms that the CiM-enabled code functions correctly and ensures that any errors observed during our fault injection campaigns for reliability assessment are only due to fault injection.

### 3.4.6. Simulation Setup

Table 3.3 lists our main configuration for this simulation setup. All our experiments (for both CPU and CiM) are performed assuming unprotected designs for two reasons: ❶ for fair comparisons between CPU and CiM, and ❷ to evaluate the raw performance and vulnerabilities of each architecture under identical conditions. This approach ensures that our analysis remains unbiased and focuses on the core attributes of CPU and CiM without the impact of additional protective measures. However, any protection scheme can be added to Sisyphus since it is based on the well-established gem5 simulator and can be easily evaluated.

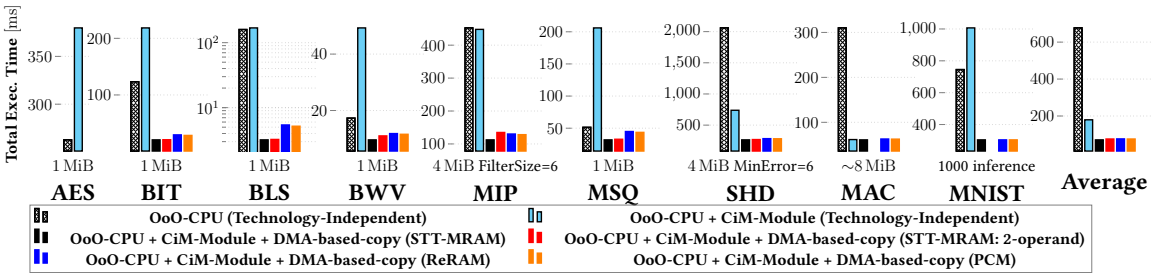



**Figure 3.4.:** Total execution time of various applications for a specified input size.

## 3.5. System-Level Evaluation

### 3.5.1. Performance and Energy Efficiency

To measure the performance and energy efficiency of CiM, we simulate multiple scenarios for various NVM technologies and report total execution time, power, and energy consumption as the evaluation metrics. Our simulation scenarios include running the application on an out-of-order CPU ❶ without any assistance from CiM, ❷ with assistance from CiM, while the commands are issued from the application layer, and ❸ with CiM assistance while storing the commands in the main memory and utilizing the DMA of CiM for data movement.

Figure 3.4 shows the total execution time for each application across various scenarios and input sizes. In scenario ❷, the majority of applications, except for a few, exhibit a preference for memory-bound operations over computation. Consequently, the anticipated performance gains from CiM in comparison to scenario ❶ are not fully realized. In scenarios ❶ and ❷, due to the large read/write latency between the CPU and main memory, which is much larger than the read/write latency of memory technologies, the total execution time for different technologies is on the order of microseconds, so we only report the results for STT-MRAM technology. Moreover, the large latency in scenario ❷ is due to the CPU handling the copying of memory rows to and from the CiM region. This process results in higher copy latency compared to scenario ❶, where the CPU takes advantage of data locality and uses its caches to perform tasks more efficiently.

In scenario ❸, by eliminating unnecessary data movement, we achieve performance improvements exceeding several orders of magnitude for most applications compared to scenario ❶. Additionally, the smaller write latency of STT-MRAM technology, compared to ReRAM/PCM, contributes to superior performance. Furthermore, **STT-MRAM: 2-op(erand)** represents the STT-MRAM technology in which the maximum number of activated rows is limited to 2. Average values are calculated based on running each application with multiple input sizes (which are not included). On average, scenario ❸ offers over a 7× latency improvement compared to scenario ❶. It is noteworthy that in this scenario, we excluded the AES encryption algorithm due to ongoing interaction between CiM and the CPU. However, custom hardware can be added to CiM to execute AES entirely within CiM, as demonstrated in [197, 198]. The differing behavior between

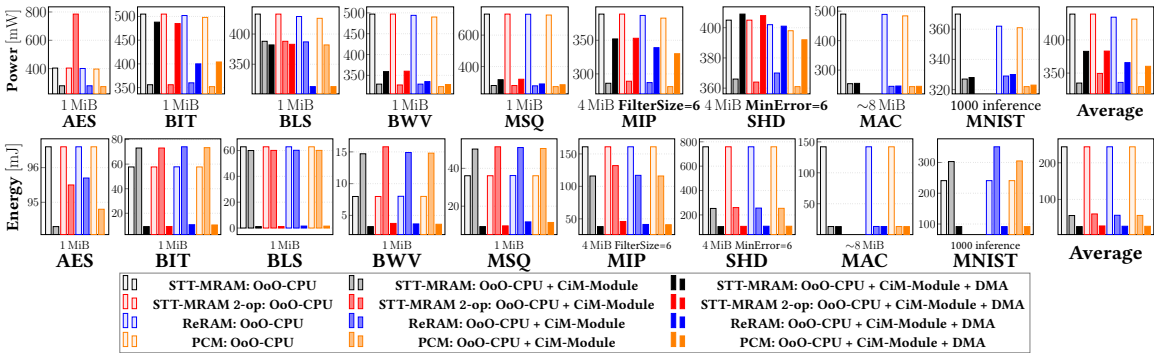



**Figure 3.5.:** Average dynamic power and energy consumption of various applications for a specified input size.

scenarios ❷ and ❸ in the MAC application (Figure 3.4) is because there are few write operations occurring. Typically, write operations are performed during the initialization stage for stationary matrices.

Figure 3.5 illustrates the average dynamic power of various components (on-chip power, including CPU and caches, memory accesses from the CPU side, and CiM+DMA power consumption), as well as energy consumption for different technologies. The power behavior depicted in this figure primarily depends on the characteristics of each application. Some applications are memory-bound, some are computation-bound, and others are a combination of both. For example, ReRAM and PCM demonstrate superior performance in memory-bound applications. Since this metric represents average dynamic power, periods in which the CPU is idle lead to reduced power consumption per application; this effect is present in all simulations in scenario ❷. By contrast, energy consumption is predominantly influenced by application execution latency and therefore increases with higher total execution times. On average, scenario ❸ exhibits nearly a 19% improvement in power and a 13× improvement in energy consumption compared to scenario ❶.

### 3.5.2. Failures in Time (FIT)

The FIT rate of a device is the number of failures that can be expected in one billion ($10^9$) device hours of operation. For each hardware structure in a heterogeneous system, including CPU microarchitectural components and CiM accelerators, a different FIT is computed using Equation 3.2. The FIT of the structure depends on three parameters: ❶ the $\text{FIT}_{\text{BIT}}$ (or raw FIT) rate, which is determined by the fabrication technology and the operational conditions (and the workload being executed in the case of CiM), which expresses the fault probability of a single bit; ❷ the number of bits of the structure; and ❸ the AVF of the structure, which is affected by the microarchitecture, the architecture (ISA), and the executed workload. The raw FIT rate expresses the number of faults that will be introduced in the component, while the AVF is the derating factor that quantifies how many of these upsets will lead to failure. The product equals the FIT rate of a particular hardware structure. For the calculation of the FIT for the CPU, we use the raw FIT rate

per bit, according to [199, 200], which is $3.85 \times 10^{-6}$ but, of course, any technology-related value can be used.

$$\text{FIT}_{\text{struct}} = \text{AVF}_{\text{struct}} \times \text{rawFIT}_{\text{bit}} \times \#\text{Bits}_{\text{struct}} \tag{3.2}$$

While the raw FIT rate (due to the underlying fault model being random alpha-particle and neutron hits) is workload-independent for CPU components, the raw FIT rate of the CiM module highly depends on the workload being executed. Different workloads activate different operations with a different number of operands that contain different data. It is thus imperative to calculate this raw FIT rate for each memory technology workload pair. To achieve this, we use a custom mode of our gem5-based fault-injection infrastructure that executes an instrumented run of the workload in question using the error model of the memory technology in question. The output of this run is the average BER of all the bits produced by the CiM module operations, i.e., AND, OR, XOR, MAC. The raw FIT rate is finally calculated using the Equation 3.3.

$$\text{rawFIT} = \text{BER} \cdot \frac{10^9 \text{ [h]}}{\text{ExecTime}_{\text{workload}} \text{ [h]}} \tag{3.3}$$

As a first quantitative comparison, we show the FIT rates for all NVM technologies (ReRAM, PCM, STT-MRAM) and the CPU configuration used in this study, as shown in Figure 3.6. The most important observation is that the CPU FIT rates, for all applications, exhibit the lowest values compared to all NVM technologies. Another important observation is that there are significant differences in the FIT rates among the different NVM technologies. As we can see, STT-MRAM memories exhibit the highest FIT rate consistently for all seven applications, while ReRAM technology exhibits the lowest FIT rates among the three NVM technologies.

Moreover, from Figure 3.6, four major insights can be extracted: ❶ ReRAM technology is the most robust technology for CiM compared to PCM, which, in turn, is more robust than STT-MRAM; ❷ both PCM and STT-MRAM provide, in some applications, virtually similar FIT (e.g., AES, BWV, MAC, and MNIST); however, in the majority of applications, their FIT difference is quite high (e.g., BIT, BLS, MIP, MSQ, SHD); this observation does not hold for ReRAM when it is compared to the other two NVM technologies; and ❸ the workload variability is extremely high. Specifically, there are significant differences among different NVM technologies regarding their sensitivity to faults. For example, comparing MIP and MSQ, although the latter is more robust than the former (i.e., the FIT rates of the latter are lower than those of the former) for ReRAM and STT-MRAM technologies, the same applications in PCM show the opposite trend (i.e., MSQ is less robust than MIP). Finally, ❹ the matrix multiplication workload that uses the MAC operation has the worst FIT out of all the applications considered. The challenges of implementing a reliable MAC operation are discussed in Section 2.5. Interestingly though, the MNIST workload, which also uses the MAC operation, has a comparatively lower FIT possibly due to the inherent approximate nature of NN inference.

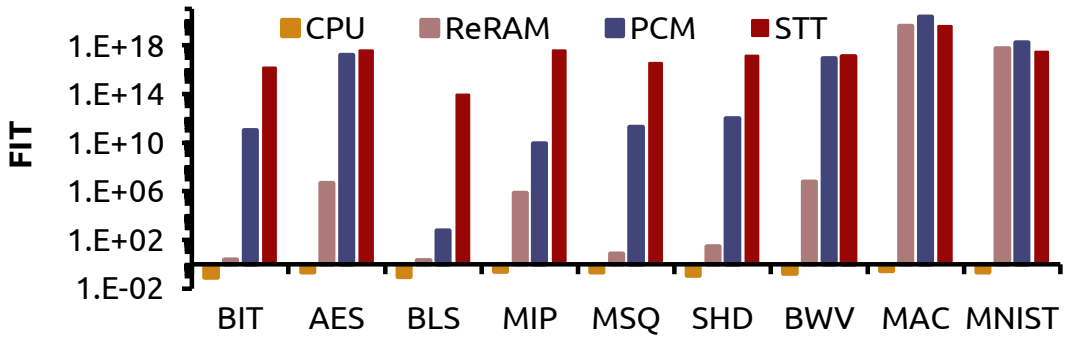



**Figure 3.6.:** FIT rates for all three NVM technologies (ReRAM, PCM, STT) and for the CPU configuration used in this study.

These observations suggest that, based exclusively on the FIT rates (i.e., the reliability-only aspect of such devices, ignoring the performance and energy consumption parameters), STT-MRAM technology is the most unreliable technology among all three NVM technologies considered in this study. However, each NVM technology and CPU provides different performance and energy characteristics for the same application. Therefore, it would be misleading if we just compare the FIT rates of each NVM technology and, more importantly, of NVM memories and the CPU. To this end, we present in the next subsections performance-aware and power-aware metrics for each compute device to quantitatively evaluate the complex interaction of all three measures considered in this chapter.

### 3.5.3. Failure per Execution (FPE)

Different computing devices (e.g., CPU, NVM-CiM, etc.) can significantly affect the performance of the executed application. To account for these differences, system or device performance can be measured by the time required to complete a full program execution. Assume, for example, that in the AES algorithm the CPU requires 10 seconds to complete the hashing procedure (AES-CPU) and CiM requires 1 second for the same task (AES-CiM). Given that FIT rate encapsulates the failures per billion device hours, we can conclude that AES-CiM has been executed 10 times more than AES-CPU. This is not a fair comparison when we account for the same algorithm/application but different computing devices with different compute capabilities. To this end, we introduce the FPE metric, as given in Equation 3.4. Figure 3.7 shows the comparison of all three NVM technologies and the CPU configuration using this metric.

$$\text{FPE} = \text{FIT} \cdot \frac{\text{ExecTime}_{\text{workload}}}{10^9} \tag{3.4}$$

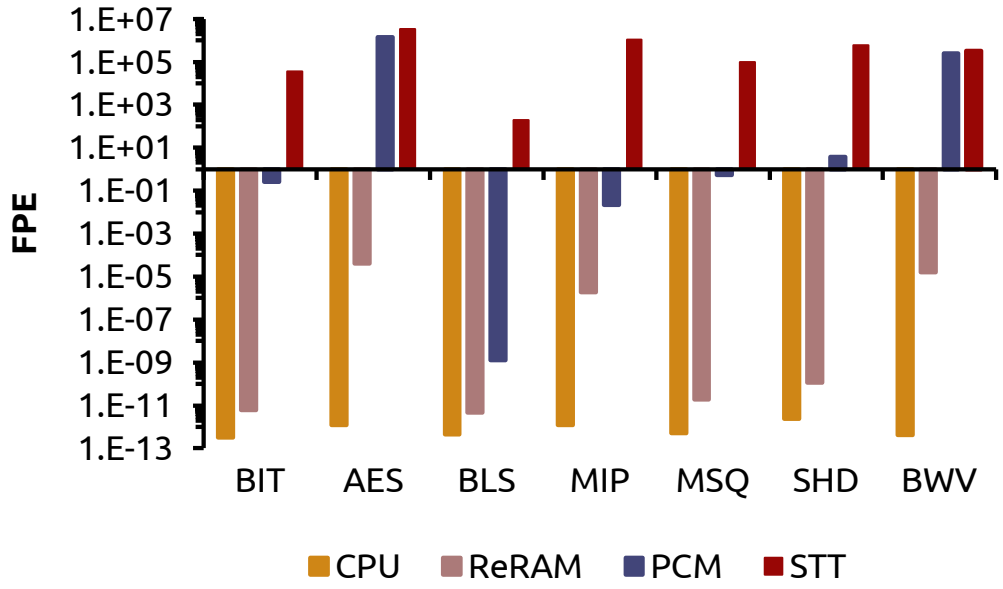



**Figure 3.7.:** FPE for all three NVM technologies and the CPU configuration used in this study.

### 3.5.4. Combined Energy and Performance Considerations

Energy consumption itself is a valuable metric that directly translates to cost, but it occasionally corresponds to very slow system configurations (very low frequencies). Such slow configurations could violate latency and throughput requirements, especially in heterogeneous environments accompanied by general-purpose CPU and CiM (or other domain-specific accelerators). To avoid this bias in the comparisons of different configurations, other metrics such as the Energy-Delay Product (EDP) (EDP $= E \times D$) and the Energy-Delay Squared Product (ED2P) (ED2P $= E \times D^2$) have been proposed for high-end systems [201]. Given that this study focuses on high-end heterogeneous systems with CPUs and CiM configurations, we have chosen to present the ED2P. This allows us to show a fair comparison for all of our experiments among benchmarks and CPU and CiM system configurations. More importantly, the ED2P metric provides a fair representation of the relation between energy and performance for different heterogeneous systems, such as general-purpose CPUs and CiM equipped with different NVM technologies.

Figure 3.8 illustrates the ED2P for all workloads across three NVM technologies and the CPU-only execution model. To enhance data visualization, we present '1/ED2P'. A larger displayed value indicates that the computing device handles the energy-performance trade-off more effectively. It is evident that, in most cases, CiM prevails. On average, the execution of workloads with CiM enabled is 14.6× more efficient than CPU-only execution, according to the ED2P metric. For three workloads (AES, MSQ, and BWV), the CPU takes the lead but with a relatively small margin. We attribute this to these applications being memory-bound, meaning that memory transfers between the CPU and CiM act as a bottleneck, preventing CiM from performing in its most efficient mode (conducting

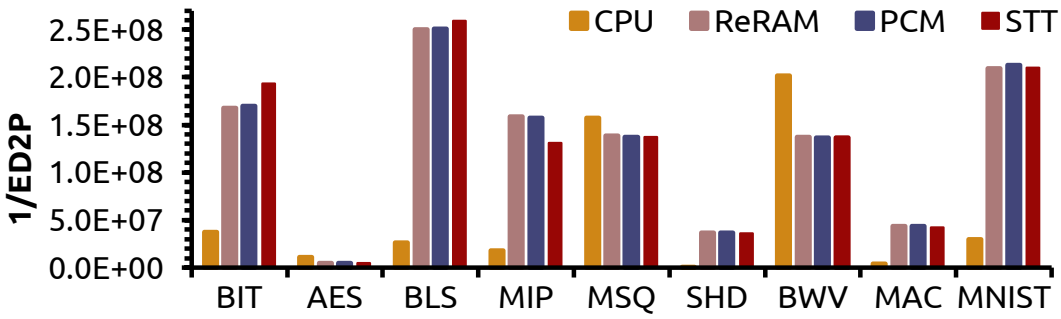



**Figure 3.8.:** Inverted ED2P (1/ED2P) for NVM technologies and the CPU configuration used in this study.

parallel dataflow operations). Among the three NVM technologies, STT-MRAM takes the lead, albeit marginally.

### 3.5.5. Putting It All Together: Performance, Energy, Reliability

To quantify the triple trade-off between performance, energy, and reliability, we employ the ERC metric described in Equation 3.5. ERC aggregates the three evaluation axes considered in this chapter. A doubling in either energy consumption or FIT results in a halving of ERC, whereas a doubling in delay results in a quartering of ERC. The greater emphasis on delay follows the rationale discussed in Section 3.5.4. Figure 3.9 presents the ERC results for all considered workloads across three NVM technologies and the CPU-only case[3]. A higher result indicates better management of the three-parameter trade-off for the specific workload-technology pair. From Figure 3.9, it is evident that the CPU-only execution model holds a distinct advantage across the entire workload spectrum, with the sole exception being the BLS application, where ReRAM takes the lead. Among the three NVM technology models, ReRAM emerges as the front-runner, only surpassed by PCM in the MIP application and STT-MRAM in the MNIST application. STT-MRAM generally lags behind the other two, primarily due to its significantly inferior resilience.

$$\mathrm{ERC} = \frac{1}{\mathrm{ED2P} \cdot \mathrm{FIT}} \tag{3.5}$$

An interesting observation pertains to the workload sensitivity of the ERC metric. For CPU evaluation, the metric remains within two orders of magnitude, whereas for CiM, it spans 20 orders of magnitude. This substantial variation is primarily due to the non-uniform resilience of CiM operations, which behave differently depending on the number of operands and the input data. Since different applications utilize CiM operations to varying extents, it is natural for ERC to fluctuate significantly. While these figures provide a high-level overview, it is important to emphasize that they apply to the current models

[3] The displayed numbers have been scaled by a uniform constant to ensure all results remain above the x-axis; the relative differences are preserved.

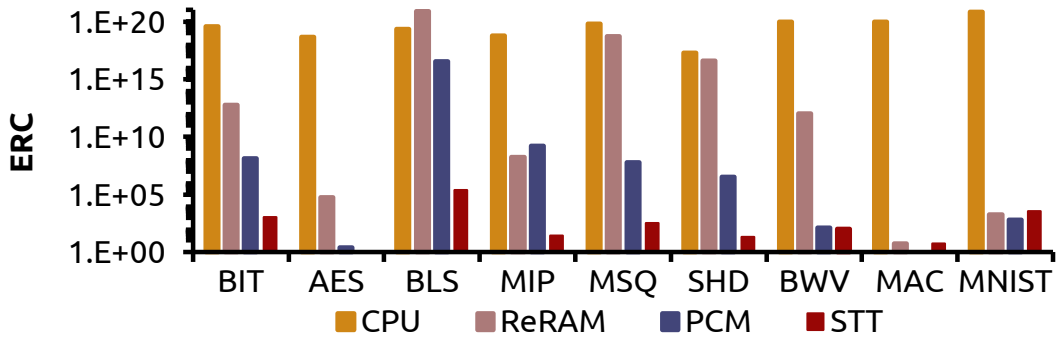



**Figure 3.9.:** ERC for all three NVM technologies and the CPU configuration used in this study.

used for the three NVM technologies and the CPU. The reliability assessment of the CPU considers only alpha-particle and neutron radiation as error sources in workload execution. As NVM-CiM technologies advance and new CPU error sources are identified, different model configurations could yield different results. However, due to the adaptable and cross-layer nature of Sisyphus, new models can be swiftly developed and integrated throughout the entire system stack.

## 3.6. Conclusion

Sisyphus is a novel, cross-layer modeling framework that evaluates system architectures combining general-purpose CPUs with NVM-based CiM accelerators. It assesses the interaction between performance, power, and resilience by utilizing microarchitectural modeling in gem5 and accurate models derived from data at the technology and circuit layers. Using Sisyphus, we revealed trends and differences among PCM, ReRAM, and STT-MRAM in terms of execution time, power consumption, and resilience. Overall, Sisyphus provides a flexible basis for exploring heterogeneous CPU–CiM systems and for guiding design choices across the device, circuit, and architecture stack.

Building on this foundation, the next chapter develops a cross-layer reliability mechanism for scouting-logic-based NVM-CiM operations. It leverages the insights and models introduced here to design and evaluate a practical scheme that improves NVM-CiM robustness while preserving its performance and energy advantages over CPU-only execution.

## 3.7. Extras

The work presented in this chapter forms the foundation of a broader research agenda on NVM-based CiM architectures and tools. Building on Sisyphus and the insights gained from our cross-layer evaluation, we have co-authored several follow-up publications in collaboration with other research groups. These works refine individual components of the framework (e.g., decoders and sensing circuitry), advance the software toolchain for CiM,

and publish Sisyphus as a reusable community resource. Below, we briefly summarize these related publications and how they extend the contributions of this chapter.

[5] **"Hardware and Software Co-Design for Optimized Decoding Schemes and Application Mapping in NVM Compute-in-Memory Architectures"**:
In this paper, we analyze and compare different decoder designs for NVM-CiM architectures and develop a heuristic compilation algorithm that ❶ allocates memory rows in a manner compatible with the chosen decoder specifications and ❷ translates decoder-agnostic CiM instructions into decoder-specific instruction sequences, thereby bridging the gap between high-level CiM abstractions and concrete hardware realizations.

[6] **"Temporal Reference Scouting Logic for PVT Reliable Logic Computation-in-Memory"**:
This work proposes a multi-stage, capacitor-based calibration scheme for NVM-CiM structures that compensates device variability and tracks PVT drift, thereby reducing bit-error rates. We further evaluate the system-level overhead of the proposed calibration mechanism and contrast it with conventional single-stage sensing circuitry.

[7] **"SHERLOCK: Scheduling Efficient and Reliable Bulk Bitwise Operations in NVMs"**:
In this paper, we automate CiM code conversion and selectively offload CiM-friendly, data-intensive parts of application kernels to CiM units, enabling a compiler-assisted CiM workflow that complements the manual CiM programming model described in this chapter.

[8] ***Modeling and Simulating Emerging Memory Technologies: A Tutorial***:
In this article, we release the framework as a publicly available open-source project, emphasize its reusability and extensibility, and provide comprehensive guidance and practical examples to help researchers and adapt Sisyphus to their own workloads and system configurations.

# 4. Cross-layer Solution for Reliable NVM-CiM Realization

**Literature**
This chapter is based on the following authored publication:
"Trust, but Verify: Reliable Compute-in-Memory via Double-Reference Sensing and Selective Recompute"

In Chapter 3, we discussed our framework for assessing NVM-CiM architectures, focusing on scouting-logic-based [152] NVM-CiM operations and evaluating *embarrassingly parallel* workloads [188]. However, we have not yet addressed the reliability challenges discussed in Section 2.5, which still hinder the practical large-scale adoption of NVM-CiM [139, 140, 141, 142, 143, 144, 174, 202]. Our evaluations show that the circuit-level decision failure rate for scouting logic, i.e., the probability of incorrect Boolean outputs, ranges from $10^{-7}$ to $10^{-3}$, which is insufficient for reliable NVM-CiM operation.

To address these limitations, we propose a cross-layer software–hardware co-design solution that improves NVM-CiM reliability while maintaining efficiency. The key idea is to identify potential circuit-level decision failures and correct them through recomputation in the CPU. To this end, we replace single-reference sensing with a double-reference scheme that identifies outputs that cannot be resolved deterministically due to overlapping distributions. This small subset of outputs is then offloaded to the CPU for recomputation.

Our experimental results show that the proposed solution significantly improves reliability, achieving an application-level decision failure rate below $10^{-12}$, while preserving the advantages of NVM-CiM over CPU-only systems in performance and energy efficiency. In particular, it delivers performance and energy-efficiency improvements of 45% and 30%, respectively, while incurring only about 1% verification overhead.

Our key contributions are as follows:

1. A double-reference sensing mechanism, including the required circuit-, architecture-, and application-level modifications.
2. An optimization approach for the double-reference design that maximizes reliability while minimizing CPU recomputation.
3. A full-system fault modeling and evaluation demonstrating the cross-layer reliability benefits of the proposed solution.

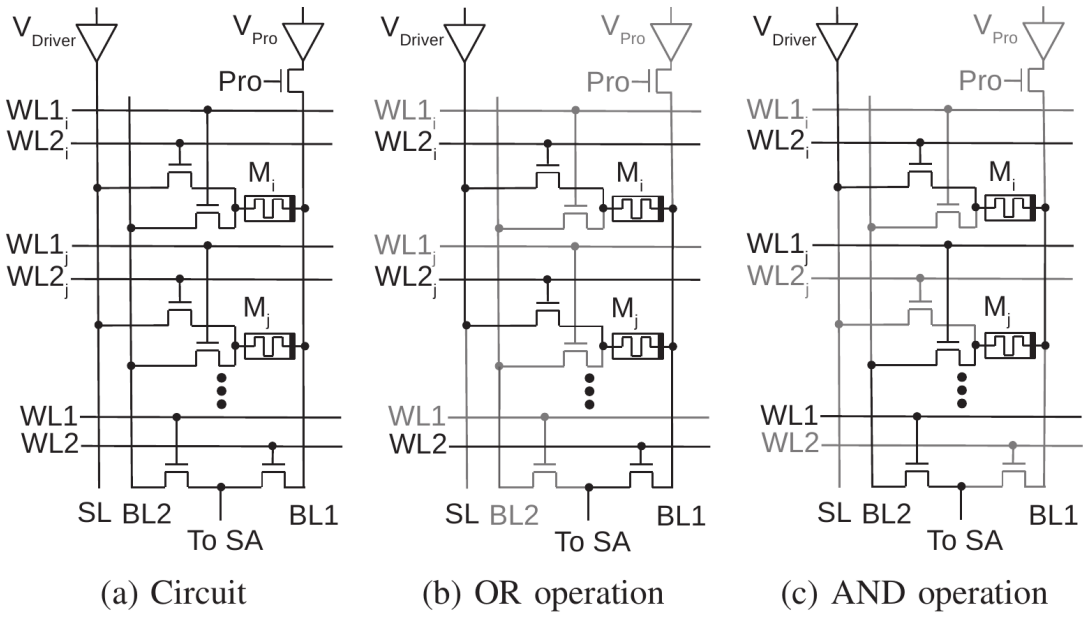



**Figure 4.1.:** Prior multi-path/reference sensing approach for CiM operations [211]

## 4.1. Related Work

The double-referencing concept has been widely used to improve the reliability of conventional memory read operations, ranging from DRAM [203] to NVM technologies [204, 205, 206, 207, 208, 209, 210]. For example, [206, 208] use two crossbar columns as references for the SA, while [209, 210] introduce *dual referencing* that combines voltage- and time-domain sensing using capacitance-based cross-coupled SAs. Such techniques enable highly accurate sensing, with [207] reporting a BER of $6.08 \times 10^{-8}$ for standard NVM reads. However, these approaches target conventional NVM memory reads rather than CiM and do not provide error detection or correction.

In the context of NVM-CiM, only a few prior works exist. For instance, [211, 212] add an extra transistor per cell, an additional WL per row, and an extra BL per column, enabling separate sensing paths for AND and OR operations (Figure 4.1). In contrast, our approach avoids distinct sensing paths and the associated area overhead.

Other works focus on test generation or offline testing for memory and CiM operations [213, 214, 215, 216, 217, 218], but none provide runtime detection or correction. At the compiler level, [7] assumes error-free two-operand operations and evaluates only application-level failure rates, while [219] generates CiM code using two-operand NOR operations to reduce errors. However, these approaches treat reliability only as an optimization objective rather than actively improving it.

## 4.2. Methodology

This section presents a cross-layer software–hardware co-design methodology to improve the reliability of NVM-CiM systems while preserving computational efficiency. The approach detects potential decision failures during scouting-logic Boolean operations and selectively recomputes suspicious outputs on the CPU, thereby avoiding full workload re-execution. Since CPUs are also susceptible to soft errors, their reliability is evaluated through microarchitectural fault injection [220], focusing on transient faults in on-chip memory arrays.

As discussed in Section 2.5, decision failures arise from overlap between BL resistance distributions, where resistance values cannot be unambiguously mapped to a specific output. The proposed method therefore targets NVM-CiM operations whose resistance values fall within this ambiguous region.

In the following, we first describe the required circuit-, architecture-, and application-level modifications that enable double-reference sensing and selective recomputation. We then introduce the corresponding fault models and baseline CPU reliability evaluation, which together provide the foundation for the subsequent system-level analysis.

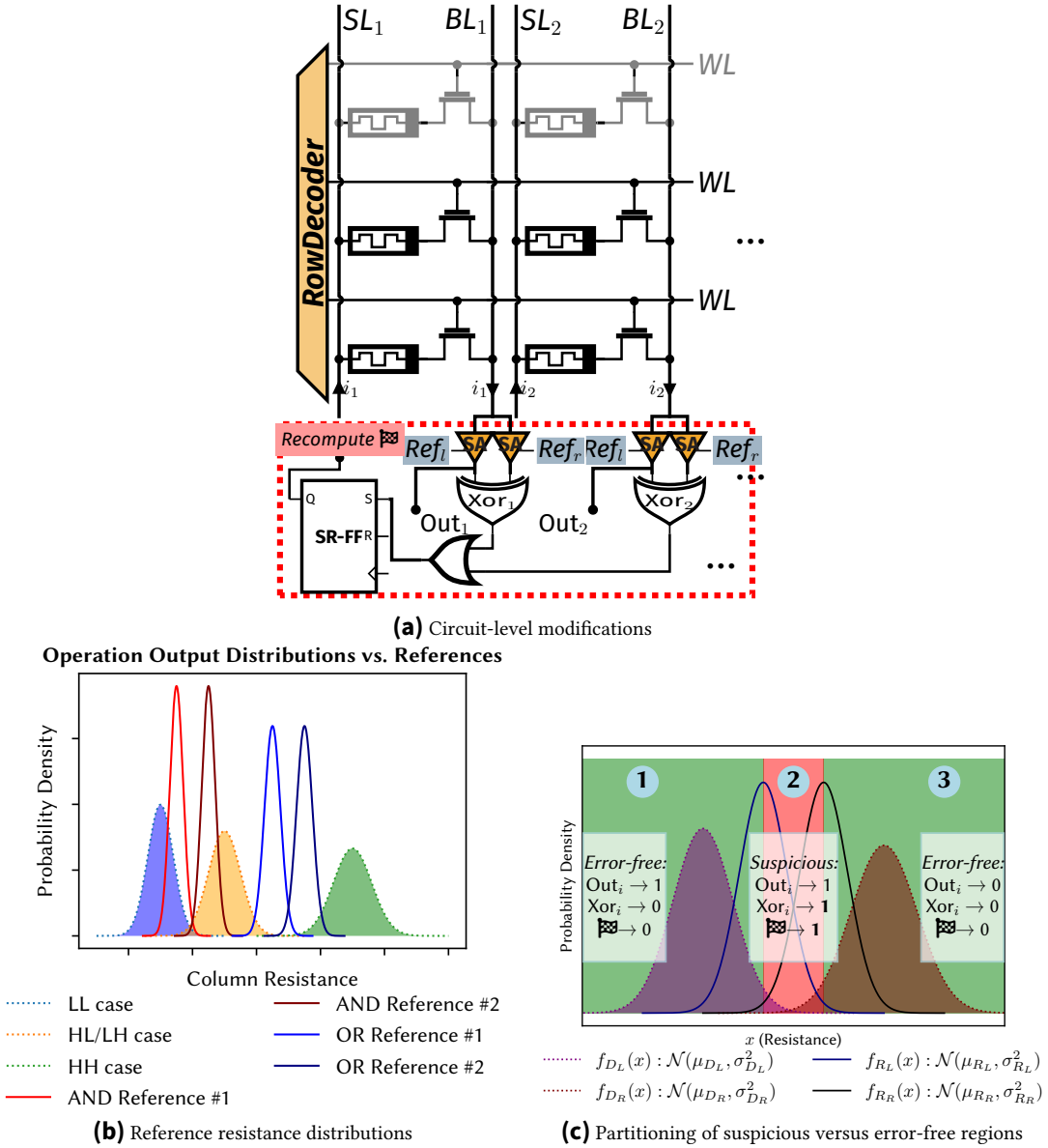
### 4.2.1. Required Modifications

#### 4.2.1.1. Circuit-Level Sensing Enhancements

To reliably detect ambiguous sensing events, the proposed design duplicates the SA for each BL. Instead of using a single reference resistance, two intentionally offset reference values partition the ambiguous region between resistance distributions and identify measurements that fall between them. Figure 4.2 illustrates the required circuit modifications and the resulting resistance distributions.

The outputs of the two SAs are jointly evaluated to assess sensing reliability. If both SAs produce identical outputs (corresponding to BL resistance values within region ❶ for the left-side distribution ($D_L$) or region ❸ for the right-side distribution ($D_R$), as illustrated in Figure 4.2c) the result is considered reliable. Conversely, a mismatch indicates a suspicious (aka error-prone) operation, occurring when BL resistance lies within region ❷. In such cases, execution proceeds using the potentially incorrect result, while a detection flag is raised for the corresponding memory region. Rarely, the tail of BL resistance distributions may fall within region ❸ for the left-side distribution or region ❶ for the right-side distribution; under these conditions, the failure is undetectable, as the output is incorrectly classified as reliable.

The trade-off involves the choice of reference values: positioning references farther apart (widening region ❷) increases the decision failure rate and the number of suspicious operations, thereby raising the CPU correction workload, while simultaneously reducing




**(a)** Circuit-level modifications

**(b)** Reference resistance distributions

**(c)** Partitioning of suspicious versus error-free regions

**Figure 4.2.:** Circuit-level modifications, reference resistance distributions, and partitioning of suspicious (❷) versus error-free (❶ & ❸) regions for the proposed double-reference scheme.

undetectable failures significantly. Because these decision failures are deterministic, re-executing the same operation on the NVM-CiM substrate would likely reproduce the same error. Moreover, selective re-execution within NVM-CiM would require a complex controller to manage parallelism, avoid deadlocks, and selectively reissue operations. For these reasons, error correction is deferred to the CPU, where selective re-execution of a small subset of inputs is simpler and more efficient.

#### 4.2.1.2. Architectural Support for Failure Detection

Since the CPU operates on 64-bit data words, a single detection flag bit is associated with each word column. As shown in Figure 4.2a, this flag is generated during NVM-CiM execution and stored in an SR flip-flop. Once set for a memory word, it remains asserted for the rest of execution. Detection flags are then used to guide selective re-execution on the CPU.

The baseline NVM-CiM architecture remains unchanged; however, detection flags must be accessible through dedicated memory addresses. After NVM-CiM execution completes, both the computed results and their associated detection flags are transferred to the CPU.

#### 4.2.1.3. Application-Level Selective Re-Execution

The proposed methodology requires two versions of the application: one targeting the NVM-CiM substrate and one targeting the CPU. Execution proceeds in two phases. First, the NVM-CiM module produces the primary outputs while generating detection flags at the micro-architectural level, without any awareness by the application itself. Next, these flags are transferred to the CPU, which re-executes only the inputs corresponding to asserted flags using the CPU version of the application. The outputs from this selective re-execution overwrite the potentially erroneous values produced during NVM-CiM execution.

### 4.2.2. Fault Modeling

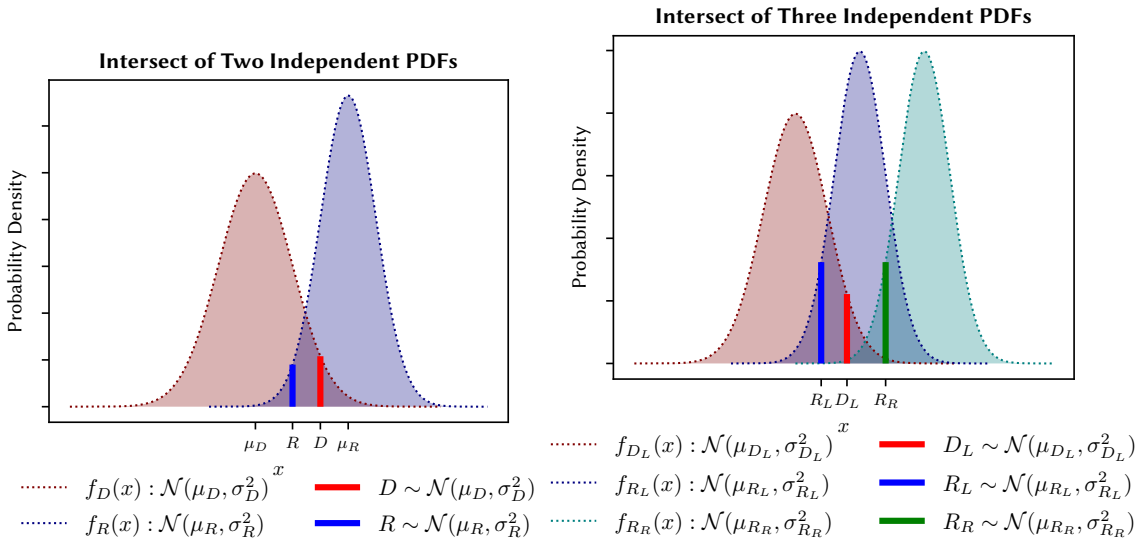
Considering Figure 4.3a, let $D \sim \mathcal{N}(\mu_D, \sigma_D^2)$ and $R \sim \mathcal{N}(\mu_R, \sigma_R^2)$ be two independent normal random variables, each belonging to a different Probability Density Function (PDF). The probability $\mathrm{P}(D{>}R)$ is given by Equation 4.1. Here, $\Phi$ denotes the Cumulative Distribution Function (CDF) of the standard normal random variable (i.e., $\mathcal{N}(0,1)$), as defined in Equation 4.2.

$$\mathrm{P}(D{>}R) = \Phi\left(\frac{\mu_D - \mu_R}{\sqrt{\sigma_D^2 + \sigma_R^2}}\right) \tag{4.1}$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,\mathrm{d}t \tag{4.2}$$

In our case, $D$ represents the BL resistance and $R$ the reference resistance of the SA. Since $D$ is expected to be smaller than $R$ when $\mu_D{<}\mu_R$, the probability of a decision failure at the SA output, i.e., $\mathrm{P}(D{>}R)$, is given by Equation 4.1. This defines the data-dependent decision failure for a conventional single-reference SA.

For the double-reference case, we calculate the data-dependent decision failure for both references. Considering Figure 4.3b and the left-side PDF for $D_L$ (a similar procedure




**(a)** Overlap region between two independent normal distributions **(b)** Overlap region among three independent normal distributions

**Figure 4.3.:** Overlap region between two independent normal distributions and region among three independent normal distributions.

applies to the right-side PDF for $D_R$), a correct output occurs when $D_L$<$R_L$ and $D_L$<$R_R$ (region ❶ for $D_L$ or region ❸ for $D_R$, see Figure 4.2c). A suspicious output occurs when $D_L$>$R_L$ but $D_L$<$R_R$ (region ❷ for either $D_L$ or $D_R$), and an undetectable output occurs when $D_L$>$R_R$ (region ❸ for $D_L$ or region ❶ for $D_R$). Note that there is no closed-form expression for P($R_L$<$D_L$<$R_R$) corresponding to suspicious outputs, as it involves three independent variables; it is typically evaluated using existing numerical integration techniques [221], as done in our simulations. Moreover, since P($R_L$<$R_R$<$D_L$) is a subset of P($R_R$<$D_L$), we use a two-variable formulation to simplify the calculation of undetectable decision failures.

We define undetectable decision failure probability ($\mathcal{F}_{\text{cim}}$) for each NVM-CiM operation as the maximum undetectable decision failure probability derived from the left-side ($D_L$) and right-side ($D_R$) PDFs, as expressed in Equation 4.3. In Equation 4.3, $\kappa$ denotes the fraction of undetectable decision failures that are not masked during application execution and may propagate to the final output. Based on our workload analysis under worst-case conditions, we assume $\kappa = 10^{-3}$. Finally, Equation 4.4 defines the overall system failure probability ($\mathcal{F}$), which accounts for both the CPU side ($\mathcal{F}_{\text{CPU}}$, detailed in Sec. 4.2.3) and the NVM-CiM side.

$$\begin{aligned}\mathcal{F}_{\text{CiM}} &= \text{P}(\text{error} \mid \text{considered reliable}) \\ &= \kappa \cdot \max\big(\text{P}(D_L > R_R),\ \text{P}(D_R < R_L)\big)\end{aligned} \tag{4.3}$$

$$\mathcal{F} = \mathcal{F}_{\text{CiM}} + \text{P}(\text{flagged as suspicious}) \times \mathcal{F}_{\text{CPU}} \tag{4.4}$$

### 4.2.3. CPU Baseline Reliability Evaluation

The operations sent to CPU for recomputation are still subject to soft errors. CPU reliability evaluation is performed using a microarchitecture-level fault injection framework built on top of the gem5 simulator. Microarchitectural simulation enables cycle-level full-system execution while modeling detailed CPU structures such as pipelines, caches, queues, and register files, allowing realistic study of fault activation and propagation. Compared to RTL simulation, this abstraction level enables large statistical campaigns and long-running workloads while maintaining sufficient fidelity for resilience studies. The infrastructure extends the gem5-MARVEL framework [220], adding support for additional CPU components and evaluation flows required by the CiM–CPU co-design explored in this chapter.

The framework supports statistical injection of transient bit-flip faults across multiple CPU microarchitectural structures, including caches, physical register files, load/store queues, the Reorder Buffer (ROB), Instruction Queue (IQ), free lists, and rename maps. Faults are injected using configurable masks that define the injection timing, location, and bit position, enabling both randomized campaigns and directed experiments. Each injected fault is evaluated through full-system simulation, capturing natural masking effects across hardware and software layers and providing realistic reliability estimates.

Fault injection campaigns consist of multiple independent runs, each introducing a single transient fault and recording its effect on execution. Outcomes are classified into masked executions, crashes, and SDCs, enabling statistical estimation of CPU error rates. These results are incorporated into the overall methodology to compare against CiM-based computation, treating the CPU as a statistically stronger execution substrate rather than an error-free oracle.

## 4.3. Results

### 4.3.1. Simulation Setup

To evaluate the proposed method, we used the *Sisyphus* framework (detailed in Chapter 3), built on top of the gem5 full-system simulator. It provides the required modifications and APIs for simulating NVM-CiM systems with suitable workloads. Since our focus is on in-memory NVM-CiM, we replaced Sisyphus's default memory controller and main memory model with NVMain [222], which enables detailed simulation of main memory behavior, particularly for NVM technologies. This is an upgrade to our previous memory model to further accurately perform NVM main memory interactions.

Additionally, we extended the NVMain configuration files to model a commercially available 1 Gbit double data rate fourth-generation (DDR4) Everspin memory module [223]. The module is based on STT-MRAM, currently the most mature NVM technology, while also exhibiting one of the worst HRS-to-LRS ratios among alternative NVMs. Demonstrating

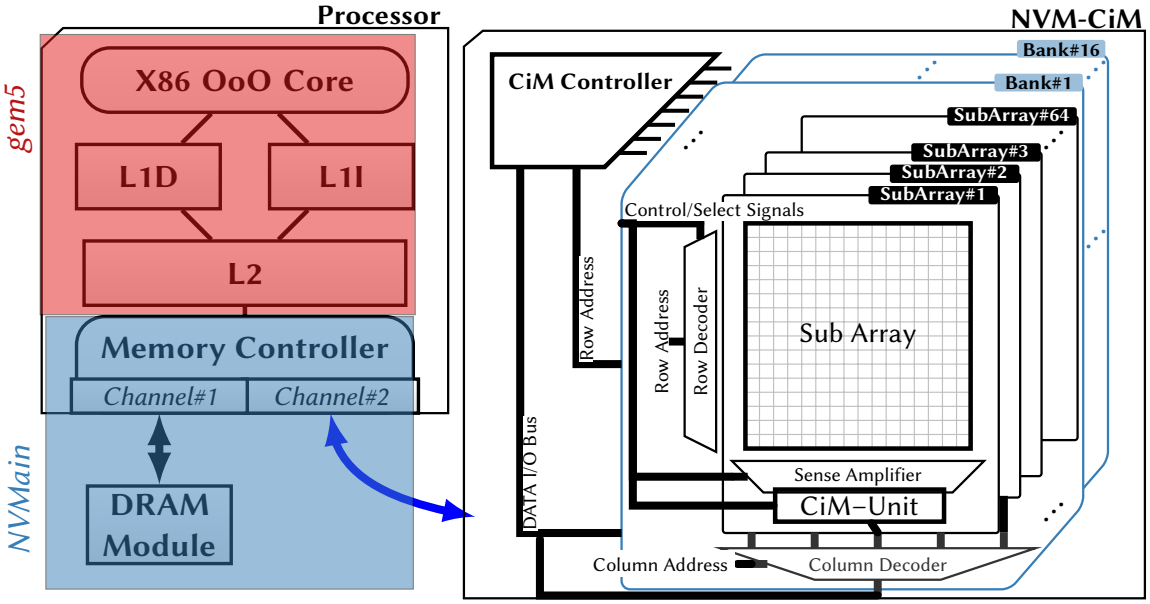



**Figure 4.4.:** System-level components and their organization for NVM-CiM setup.

the effectiveness of our method on this technology highlights its practicality and suggests applicability to other technologies with more favorable HRS-to-LRS ratios.

Figure 4.4 shows a simplified system-level configuration of our NVM-CiM. By default, *Channel#2* is disconnected, and no NVM-CiM memory is present in the CPU-only configuration. We use *Channel#2* to separate the physical address spaces of NVM-CiM and DRAM-based main memory. As illustrated in Figure 4.4, the configuration consists of a single NVM-CiM chip without data interleaving, comprising a CiM controller and multiple NVM-CiM subarrays. The CiM controller is similar to a DRAM chip controller but employs a distinct state machine and dedicated circuitry to orchestrate CiM operations. Table 4.1 summarizes the key simulation parameters.

To develop an accurate system-level-compatible circuit model, we construct a statistical model of a single bit-cell consisting of an STT-MRAM device in series with an N-type Metal-Oxide-Semiconductor (NMOS) transistor, with parameters listed in Table 4.1. For this purpose, $10^4$ Monte Carlo simulations are performed per bit-cell using Cadence Virtuoso, with the 22 nm FD-SOI library from GlobalFoundries. The STT-MRAM employs a Perpendicular Magnetic Anisotropy (PMA) configuration [224], with parameters listed in Table 4.1. Since the resulting behavior follows a normal distribution, we extract the mean ($\mu$) and standard deviation ($\sigma$). These parameters are then used in a Python-based analysis to perform $10^8$ Monte Carlo simulations per cell-state combination, modeling BL resistance under multi-row activation. A similar procedure is used to model variations in the trim circuitry for reference generation.

Finally, the extracted parameters are incorporated into the fault model to compute the decision failure probability according to Equation 4.1. Table 4.2 reports the decision failure probability for 2-operand OR operation. These values are subsequently used in the full-system simulations for fault modeling. For the proposed NVM-CiM, three cases

**Table 4.1.:** Simulation-setup parameters/configurations.

| | | |
|---|---|---|
| **Circuit-Level Parameters** | | |
| Oxide Layer Thickness | 0.85 nm | |
| Free Layer Thickness | 0.70 nm | |
| MTJ Surface (circle) | r=40 nm | |
| $TMR_0$ (TMR at 0 V) | ~200% | |
| Resistance-Area Product (RA) | 10 Ω µm$^2$ | |
| LRS ($3\sigma$) | 3.1% | |
| HRS ($3\sigma$) | 8% | |
| 22 nm FD-SOI library (GlobalFoundries) | | |
| **Extracted Circuit-Level Parameters** | | |
| | $\mu$ | $\sigma$ |
| HRS + NMOS | 23 kΩ | 1000 Ω |
| LRS + NMOS | 12 kΩ | 500 Ω |
| **System-Level gem5 Config file** | | |
| CPU type | Out-of-Order | |
| ISA / Clock Frequency | x86–64 / 2 GHz | |
| Compiler & Optimization | g++ -O3 | |
| Load / Store Queue Entries | 32 / 32 | |
| Issue Queue Entries | 64 | |
| ROB Entries | 128 | |
| Physical Registers (Int / FP / Vec) | 128 / 128 / 128 | |
| L1 Instruction / Data Size | 32 / 64 KiB | |
| L1 Latency (Tag / Data / Resp.) | 2 / 2 / 2 cycles | |
| L2 Size | 2 MiB | |
| L2 Latency (Tag / Data / Resp.) | 20 / 20 / 20 cycles | |

**Table 4.2.:** Data-dependent decision failure rates for OR2 operation.

| Method | Error-Type | Actual Output: '1' | Actual Output: '0' |
|---|---|---|---|
| Conventional NVM-CiM | Erroneous | $1.64 \times 10^{-7}$ | $1.18 \times 10^{-8}$ |
| | Error-free | 1-$1.64 \times 10^{-7}$ | 1-$1.18 \times 10^{-8}$ |
| **Proposed NVM-CiM** | Suspicious | $9.79 \times 10^{-7}$ | $7.87 \times 10^{-7}$ |
| | Undetectable | $3.86 \times 10^{-10}$ | $2.55 \times 10^{-9}$ |
| | Error-Free | ≈1-$10^{-6}$ | ≈1-$10^{-6}$ |

are considered: in the suspicious case, a wrong Boolean output is generated along with a detection flag; in the undetectable case, a wrong Boolean output is generated without setting a detection flag; and in the error-free case, a correct output is generated without setting a detection flag.

### 4.3.2. Simulation Results

To improve reliability at the application level, all 4- and 8-operand Boolean operations were decomposed into 2-operand primitives. Furthermore, due to the high decision failure rate of the 2-operand AND operation, these operations were reformulated using OR and NOT equivalents. Although this transformation increases the number of operations and thus introduces some performance degradation, the reduction in suspicious verification candidates leads to only minor overall overhead. It also reduces the required reference generation circuitry from four to two.

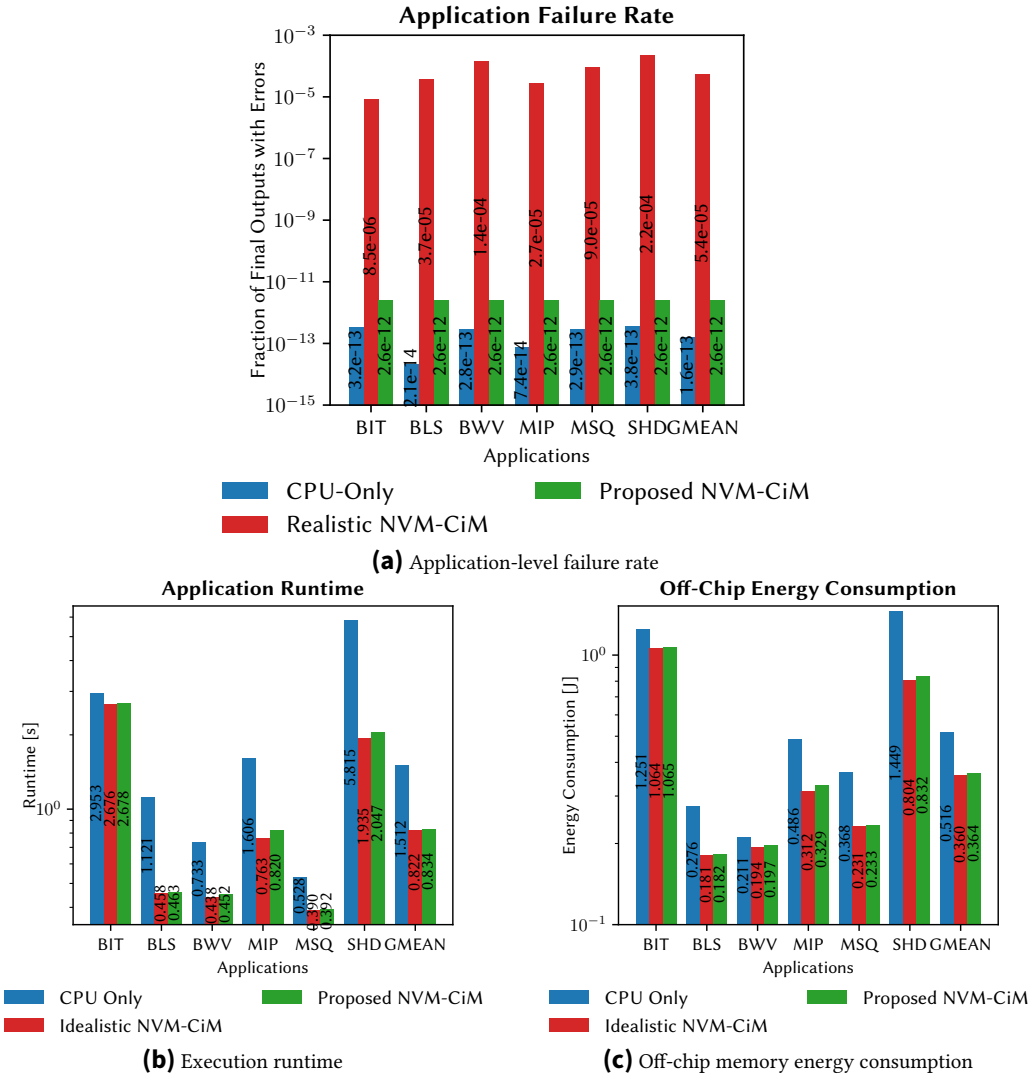
Figure 4.5 shows the simulation results for the configuration described in Tables 4.1 and 4.2, using a 2 GHz CPU clock, $\Delta\text{Ref}_L$=−100 Ω, and $\Delta\text{Ref}_R$=300 Ω for the OR references. Figure 4.5a shows the fraction of faulty final outputs for the realistic NVM-CiM, obtained by comparing the final outputs with the golden results. For the proposed NVM-CiM, no final errors occur because all suspicious outputs are recomputed; therefore, we report $\mathcal{F}$. Overall, the application-level failure rate for realistic NVM-CiM using only 2-operand OR operations is on the order of $10^{-5}$ without verification, whereas the proposed NVM-CiM achieves a failure rate on the order of $10^{-12}$, approaching the reliability level of the CPU-only baseline. Figure 4.5b–Figure 4.5c show runtime and off-chip memory energy consumption, respectively. On average, NVM-CiM outperforms the CPU-only baseline by 45% in performance and 30% in energy efficiency. The proposed NVM-CiM introduces only about 1% additional overhead due to verification and correction while exponentially improving reliability compared to the unverified scheme.

Because the evaluated workloads do not support multithreaded or parallel execution, we swept the CPU clock frequency from 1 GHz to 8 GHz to approximate a fair comparison between a high-performance CPU implementation and NVM-CiM. Figure 4.6 shows the resulting runtime, off-chip power, and energy trends. Since the CPU relies on DRAM-based memory, its power consumption scales with frequency differently from NVM-CiM. Frequency scaling therefore provides a best-case proxy for increased computational capability, but does not fully capture realistic cache hierarchy or parallel-execution effects.

Finally, Figure 4.7a shows the average percentage of final outputs requiring verification for different choices of $\Delta\text{Ref}_L$ and $\Delta\text{Ref}_R$. Figure 4.7b similarly presents $\mathcal{F}_{\text{cim}}$ for each $(\Delta\text{Ref}_L, \Delta\text{Ref}_R)$ combination. As observed, there is a clear trade-off between the fraction of suspicious outputs and $\mathcal{F}_{\text{cim}}$, with the optimal parameter combinations highlighted in the heatmaps. Moreover, Figure 4.7 shows that the impact of $\Delta\text{Ref}_L$ and $\Delta\text{Ref}_R$ on this trade-off is asymmetric.

## 4.4. Conclusion

In this chapter, we addressed the reliability limitations of scouting-logic-based NVM-CiM architectures, which hinder their large-scale adoption despite their potential for high-performance, energy-efficient in-memory computation. We proposed a cross-layer software–

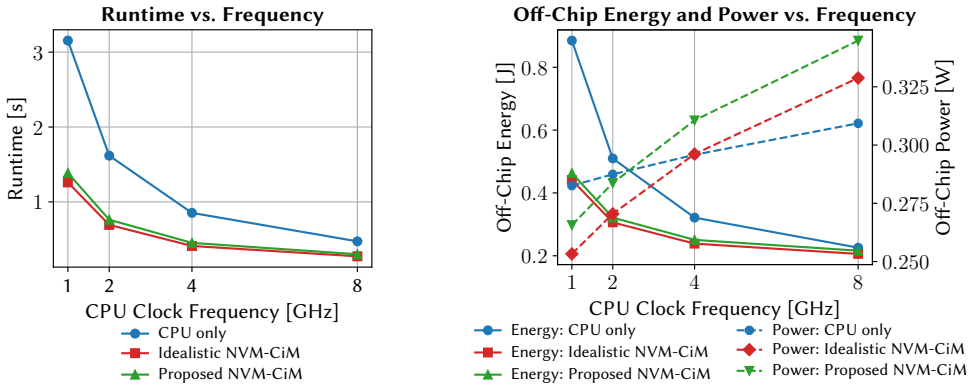



**(a)** Application-level failure rate

**(b)** Execution runtime

**(c)** Off-chip memory energy consumption

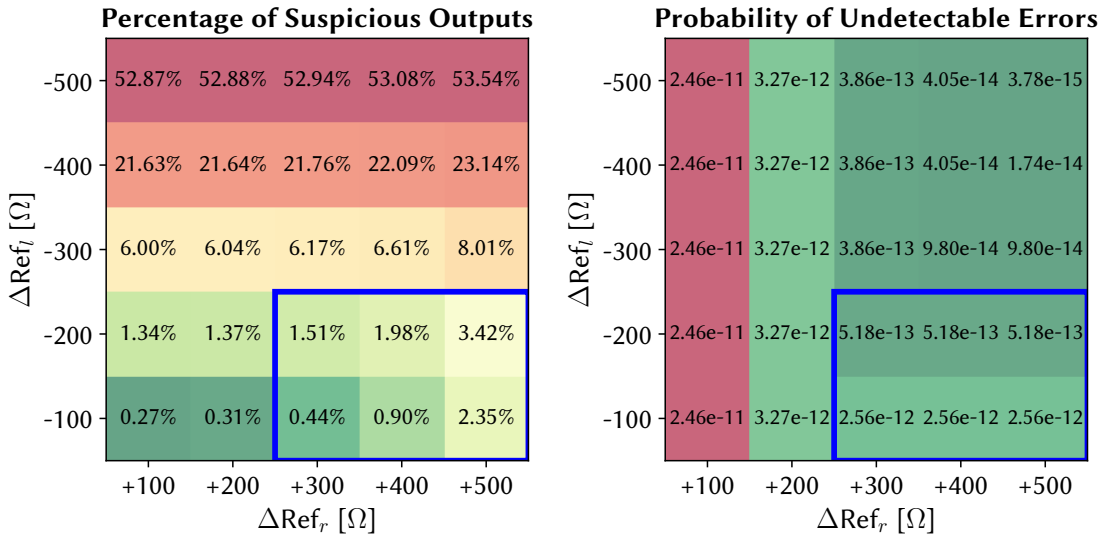
**Figure 4.5.:** Application-level failure rate, execution runtime, and off-chip memory energy consumption under a 2 GHz CPU configuration.




**(a)** Execution runtime

**(b)** Off-chip power and energy consumption

**Figure 4.6.:** Execution runtime and off-chip power and energy consumption trends for varying CPU clock frequencies.




**(a)** Proportion of outputs requiring verification

**(b)** Likelihood of undetectable errors

**Figure 4.7.:** Trade-off analysis heatmaps between the proportion of outputs requiring verification and the likelihood of undetectable errors. Optimal regions are highlighted with boxes.

hardware co-design approach that combines double-reference sensing for circuit-level detection of decision failures with application-level verification to ensure correctness. Our evaluations show that the proposed solution reduces the application-level decision failure rate to the order of $10^{-12}$, approaching the reliability level of CPU-only execution while preserving the advantages of NVM-CiM and achieving up to 45% performance improvement and 30% energy reduction.

Beyond validating the proposed cross-layer scheme in isolation, these results establish a quantitative baseline for comparing alternative in-memory compute substrates and circuit organizations. Building on this foundation, the next chapter turns to benchmarking different NVM-CAM designs for similarity computation. There, we evaluate scouting-based and CAM-based NVM-CiM flavors in a technology-to-application flow, highlighting how their latency, energy, and reliability characteristics complement the cross-layer protection techniques developed in this chapter.

# 5. Benchmarking Various NVM-CAM Designs

**Literature**
This chapter is based on the following authored publication:
"Cross-Layer Reliability Evaluation of In-Memory Similarity Computation"

*Similarity computation* (with scan/search operations as a subset) between two binary patterns can serve as a primary kernel in various computer science applications [225, 226], including web engines, social media platforms, database systems, compression techniques, file and document management systems, and more [227, 228]. Efforts to enhance the efficiency of this operation range from algorithmic optimizations [229, 230] to hardware enhancements [231, 232, 233, 234].

Besides conventional hardware platforms, Associative Memory (AM)[1] is also a hardware solution that directly aims to design specialized comparison circuitry to execute search operations in a single cycle. AMs find popularity in various applications requiring high-speed lookup tables, such as packet forwarding in network routers, memory mapping in TLB, tag checking for caches, and buffering in multiprocessor systems. Despite the increasing capacity of AMs over recent years, from a few kilobytes to several megabytes, the primary challenge in AM design remains reducing power consumption without compromising speed [72, 235, 236, 237].

To achieve the high-speed feature of the AM and energy efficiency of the NVMs, an NVM-based AM can be utilized to efficiently perform the task of similarity computation in the concept of NVM-CiM. However, NVM-CiM-based similarity computation faces reliability challenges due to process variation in NVM devices, circuit-level non-idealities, and the lower noise immunity of analog computation.

The main focus of this chapter is a comprehensive analysis of two flavors: ❶ *scouting-based* and ❷ *CAM-based*, representing different styles of CiM paradigms for similarity computation. Our analysis centers around the investigation of reliability versus energy/performance trade-off using a cross-layer approach. Systematically, we start with accurate modeling of

[1] AM is a broad, general term for any memory system that retrieves stored information based on content or partial cues rather than a specific address or index. CAM is a specific hardware implementation of this idea: it stores data in such a way that you can input a value (or part of it) and the hardware returns the address or matching entry in one (or a few) clock cycles. In other words, all CAMs are AMs, but not all AMs are CAMs [72].

the NVM device. Further, we extend our analysis to the circuit level and perform accurate SPICE simulations for each of these two flavors. To fill the gap between the circuit and application levels, we augment gem5 as a cycle-accurate architecture with hardware-aware extensions of these two flavors. Such a framework enables us to evaluate performance and energy. The abstracted error models are then used for application-level error propagation analysis, aka AVF.

Overall, our findings demonstrate significant improvements in latency, ranging from 1.2× to 3.4× and from 2.2× to 12× for both flavors compared to CPU performance. Regarding energy efficiency, the second flavor outperforms the first with improvements ranging from 5.3× to 6.7×. Reliability is primarily influenced by device technology and similarity computation flavors, with variability ranging up to 28% in terms of AVF. However, application specification plays a crucial role in this context. Importantly, latency and energy consumption are largely independent of technology. For example, despite a more than 10× difference in write latency between the two NVM technologies, this variation is mitigated by overall system interaction.

## 5.1. NVM-CiM-based Similarity Computation

The computation of similarity between binary patterns can be effectively done by the analog characteristics of NVM-CiM. In the case of scouting-based flavor, there is a generic NVM-CiM macro that performs vector-wise Boolean operations, e.g., XNOR. With the proper post-processing using standard CPU instructions, such generic NVM-CiM macro can perform the task of similarity computation. In the case of CAM-based flavor, however, there is a dedicated NVM-CiM as a similarity computation unit which also uses analog computing. These flavors have varying performance, energy consumption, and reliability characteristics, which will be further analyzed within the scope of our end-to-end technology-to-application flow.

### 5.1.1. Scouting-based Similarity Computation

The XNOR operation can be executed within the NVM-CiM using different methods, one of which involves employing the *scouting-logic* technique. This technique leverages the analog nature of the device to simultaneously activate multiple memory rows (using the address decoder), allowing for the execution of a specific Boolean operation (such as XNOR) in the analog domain by performing the comparison with a known reference. Subsequently, the result of the Boolean operation can be determined through this comparison using a one-bit comparator (realized by the SA) to generate the result. As shown in Figure 5.1, the elements of input vectors $[P_1 \dots P_n]$ and $[Q_1 \dots Q_n]$ are programmed in arrays, and

per-bit XNOR operation can be performed at the same time. To obtain the number of similar bits, a popcount operation[2] is performed on the XNOR results.

### 5.1.2. NVM-CAM Similarity Computation

A 2T2R NVM-based CAM is introduced in [119] and shown in Figure 5.2. The prototype (indicated by **P**) and the query (indicated by **Q**) patterns are applied to the 2T2R structure in a complementary fashion, i.e., **P** and **Q** in one branch and ~**P** and ~**Q** to the other one. By conventionally assuming the binary '1' as HRS and $V_{DD}$ in the prototype and query, respectively, and the other way around for the query, in the case of match (**P** = 0 (1), **Q** = 0 (1)) the already pre-charge ML has an HRS-included discharge path, while in the case of mismatch (**P** = 0 (1), **Q** = 1 (0)), the discharge path will be through LRS. This way, at the sampling time, the voltage of the ML shows a higher value in the case of a match compared to a mismatch. Therefore, the higher the number of similar bits, the higher the voltage of the ML. Since the rest of the microarchitecture is digital, the voltage of the ML needs to be digitized using an ADC. The output of the ADC correlates (meaning not exactly equal) with the number of similar bits in the **P** and **Q** patterns.

Encoding binary '1' (HRS) and '0' (LRS) in the prototype, along with high and low voltages in the query, respectively, facilitates this comparison. Utilizing this structure for the task of similarity computation [78, 238, 239, 240, 241, 242]. Notably, the smaller HRS-to-LRS ratio in NVMs compared to SRAM limits the ML's length to a certain threshold (e.g., 64-bit [78]). In the CAM-based similarity computation, to maintain the connection between this analog module and the rest of the digital computer, an ADC is required to convert the voltage of the ML to a digital corresponding.

## 5.2. Related Work

Among the two discussed flavors for similarity computation, the CAM-based structure receives greater interest in the literature [78, 238, 239, 240, 241, 242]. The existing related work in this scope has two common shortcomings: ❶ lack of holistic cross-layer analysis from device to application level, and ❷ limiting the applications-level analysis only to brain-inspired HyperDimensional Computing (HDC). Neither of the existing related works has precisely incorporated the effects of the architecture level in their analysis.

To fill the gap and conduct holistic end-to-end technology-to-application-level analysis, this chapter delves into the detailed hardware-aware implementation of these two flavors of similarity computation using the well-known gem5 full-system framework. Despite the benefits of NVM-CiM, its reliability needs careful examination. Therefore, performing hardware-level fault injection to analyze error propagation from technology and circuit to the application level becomes imperative.

[2] Population Count Operation

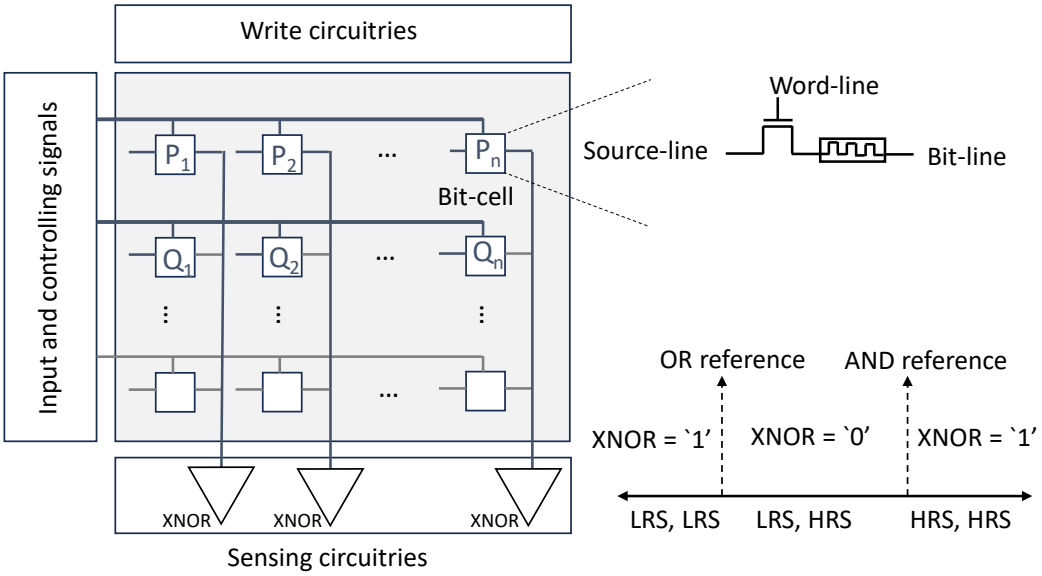



**Figure 5.1.:** Scouting-logic with two references for realizing the XNOR operation.

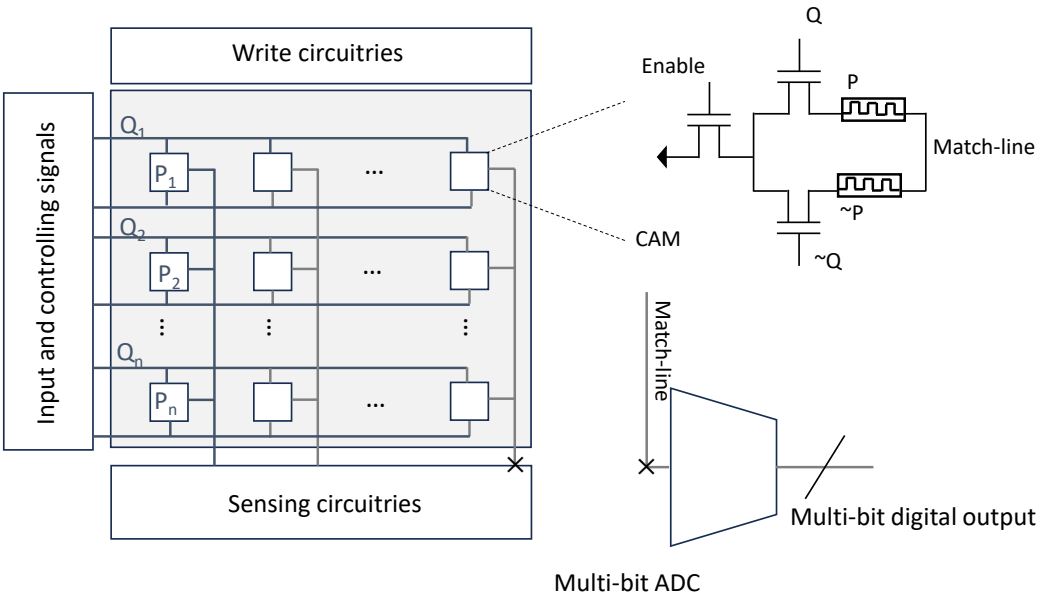



**Figure 5.2.:** CAM-based implementation of similarity assessment.

## 5.3. Technology-to-Application Analysis Flow

This section presents our comprehensive assessment of the end-to-end framework for scouting- and CAM-based approaches to similarity computation. Our end-to-end framework is organized into two primary parts: ❶ architectural-level implementation and ❷ fault modeling, abstraction, and injection. These parts are further elaborated upon in the following subsections.

### 5.3.1. Architectural-level Implementation

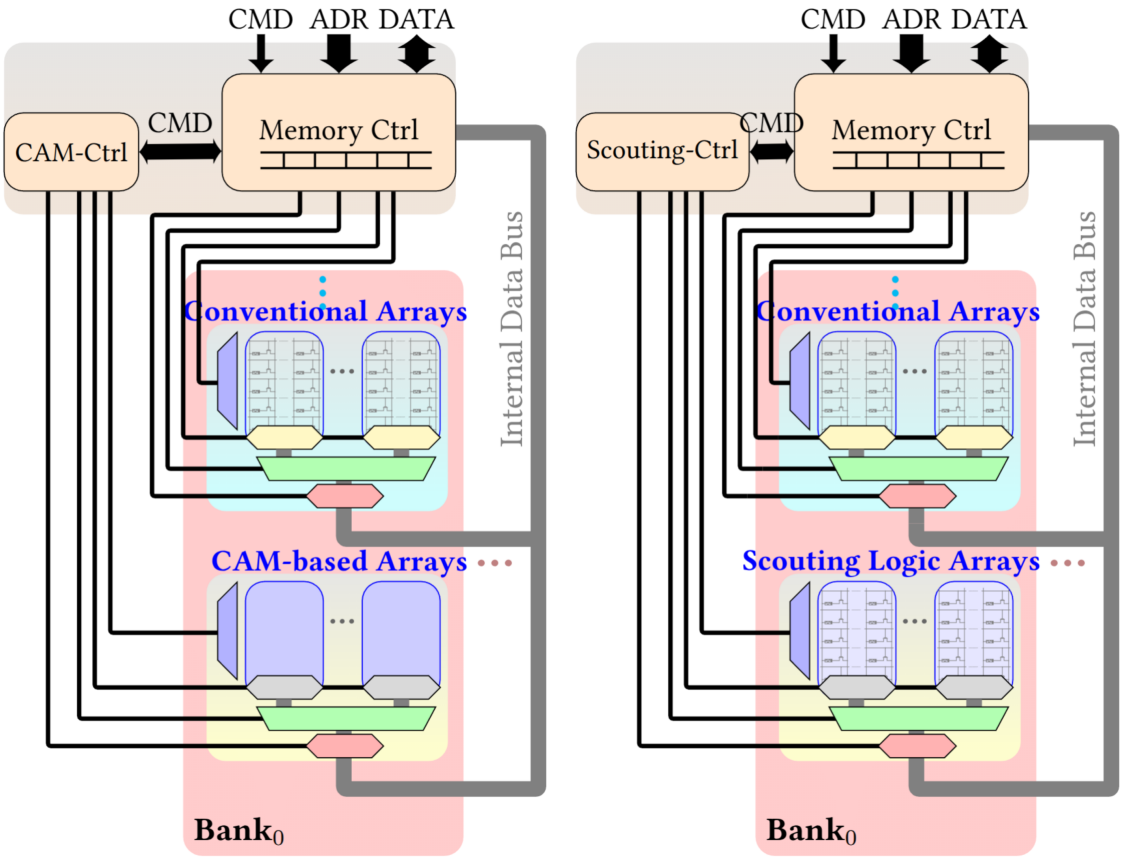
To comprehensively consider the full-system parameters in our simulation, from CPU and cache interactions down to main memory, and simultaneously consider interactions between the operating system and our main applications, we utilized the gem5 full-system framework and extended it with similarity computation CiM modules. It is a well-known simulation framework in the computer architecture community and enables us to model any device component architectural module in a cycle-accurate fashion.

Our objective was to precisely model both scouting- and CAM-based functionalities as additional extensions within our NVM-CiM architecture. To accomplish this, we extended the main memory component (including the memory controller and memory array) of gem5 to implement them. We aimed to carry out this implementation and modeling in a way that ensures our architectural-level extension is fully hardware-aware. This approach allowed us to guarantee that architectural-level fault injection aligns with the technology and circuit-level specifications.

To ensure compatibility with the operating system and minimize hardware modifications, we employed memory-mapped IO communication to access our extensions from the application layer. These extensions have dedicated physical locations and addresses, managed by a unit integrated within the memory controller. Consequently, the memory controller filters out these specific addresses and directs them to the extension's controller for handling. Moreover, to achieve bank-level parallelism, these physical locations are distributed across all the banks. Figure 5.3 illustrates these modifications in detail, with only a slight variation for the extension controller and their peripheral memory arrays.

We provide a system driver to exclusively allocate memory-mapped resources to our applications, preventing allocation by other system processes. Additionally, we offer an API library for accessing and modifying these resources. Furthermore, we integrated a separate DMA unit into our extension controller to facilitate intra-main memory data movement. By default, the CPU manages data copying between memory locations, leading to unnecessary movement between main memory, cache, and back to main memory. This inefficiency increases energy consumption and latency, contradicting the concept of CiM. While DMA-based data movement offers efficiency, it may raise concerns regarding memory access violations and cache coherency issues. For simplicity, we assume the programmer has full system control and is responsible for managing these potential issues.

**Scouting-based Scenario** Our extension executed the XNOR operation between the query and a list of data inputs. Subsequently, the intermediate preliminary results were transferred to the CPU for post-processing and generating final results depending on the application specifications. Although this off-chip operation might seem insignificant compared to the CPU's ability to perform it rapidly, from the data transmission perspective, it does not reduce the data size. In scenarios where the input size is sufficiently large, individual data elements may be loaded into the cache multiple times due to cache misses




**(a)** Scouting-based **(b)** CAM-based

**Figure 5.3.:** Main-memory modifications are implemented for both scouting- and CAM-based extensions, differing in internal circuitry for both the controller and memory periphery.

resulting from its capacity limitations. In contrast, preparing these intermediate results within the main memory with minimal energy consumption and largely in parallel enables the CPU to dedicate its resources to critical tasks, especially in a full-system environment.

**CAM-based Scenario** Its functionality was similar to the previous scenario except the intermediate preliminary results consist of 8-bit ADC values. Here, by reducing the data size of preliminary results, we enhanced latency and reduce energy consumption associated with the memory wall problem. It is worth mentioning that in this paper, we are targeting the general definition of similarity computation. Therefore, we are not referring to the classical definition of AM, which involves searching for a key within a collection of data and retrieving the corresponding index or value if it exists. Instead, we are returning a collection of similarity values. The reason for this approach is to enable the implementation of a broad set of applications for this module, rather than limiting it to exact matching scenarios. Nevertheless, the classical definition of AM can easily fit into our general definition by slightly modifying the AM hardware.

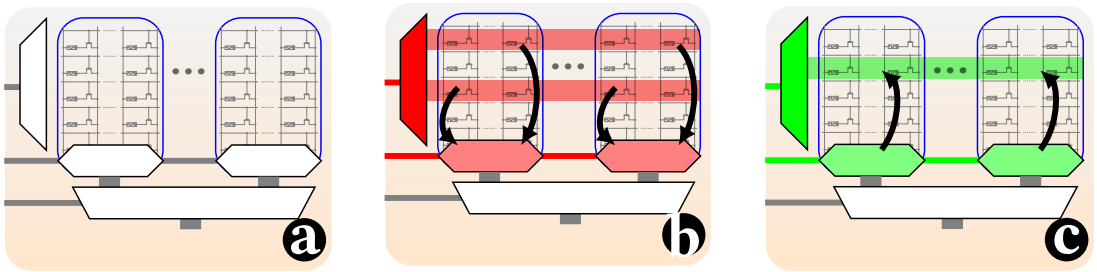



**Figure 5.4.:** XNOR-operation execution order on scouting-based extension. Two rows are activated simultaneously, and the result is then written back to the memory arrays after performing the operation.

Figure 5.4 illustrates an example of bulk XNOR operation performed using the scouting-based method. In Figure 5.4a, the idle memory state is shown. Figure 5.4b demonstrates the activation of two rows simultaneously, followed by the execution of the XNOR operation in the analog domain with the assistance of comparators. Subsequently, in Figure 5.4c, the results are written back to a particular row. These signaling processes are managed automatically by the extension controller (see Figure 5.3). A similar process can be considered for a CAM-based scenario, except they are performed sequentially for each memory block.

## 5.3.2. Fault Modeling, Injection, and Error Propagation Analysis

Our main objective in performing the fault injection was to analyze the impact of the technology- and circuit-level faults on the application-level accuracy. Besides, such fault injection analysis provides insight into the architecture- and application-level error-masking capability. In the following, we further elaborate on the exploited cross-layer fault modeling, abstraction, and injection flow. Figure 5.5 shows the flowgraph of our fault injection methodology.

### 5.3.2.1. Technology-level Fault Modeling

At this level we used the measurement-verified Verilog-A compact models of the NVM devices with the incorporation of the process variation, temperature, and time dependability (see Section 2.5). Both scouting- and CAM-based flavors only require resistance sensing, therefore, we can abstract the NVM devices as their LRS and HRS distributions obtained by the Monte Carlo simulation with SPICE tools and fitted to a known distribution. Therefore, the entity that will transferred from the technology to the circuit level is the statistical distribution of LRS and HRS for each NVM technology. The technology-level non-ideality is required for both the scouting- and CAM-based flavors and can be performed similarly.

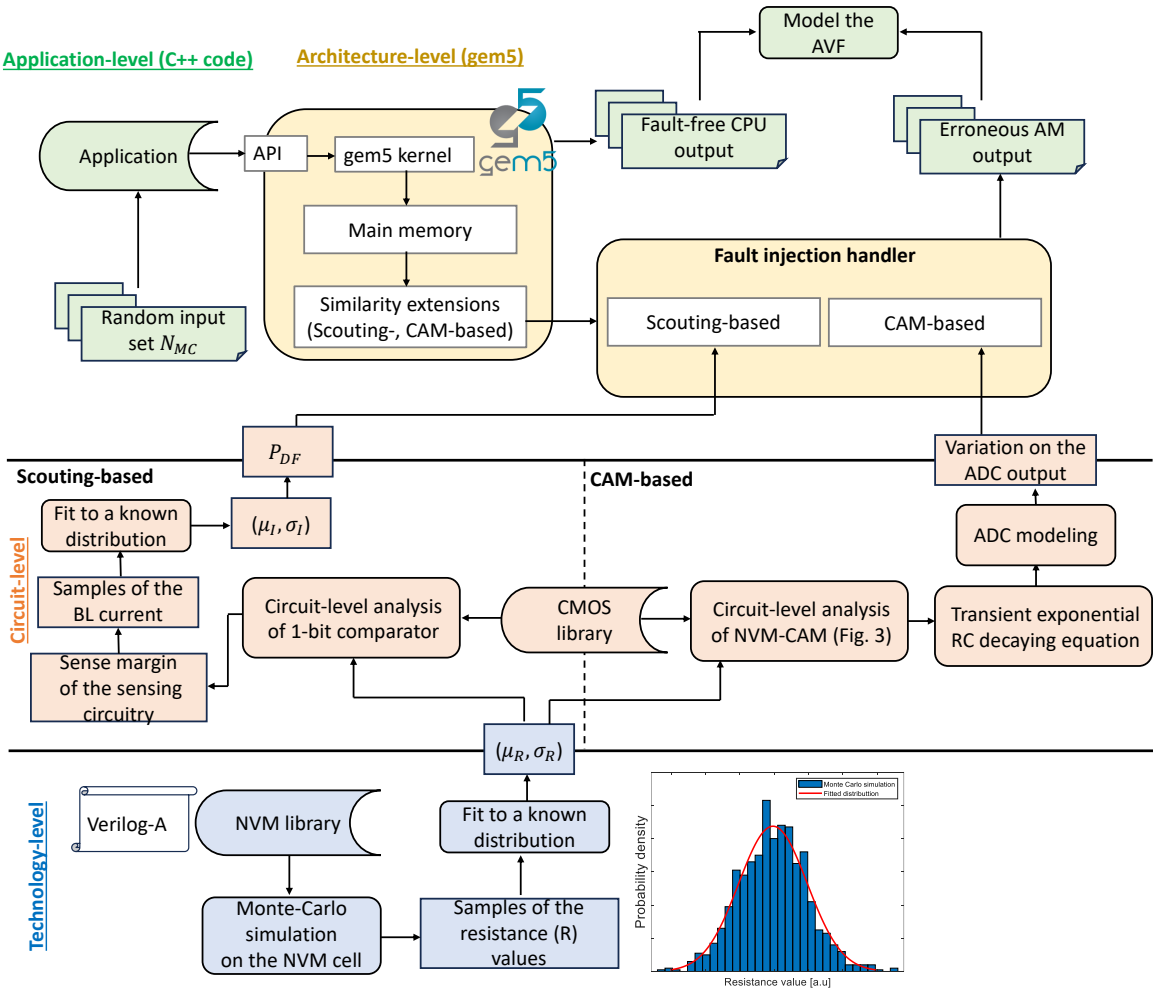


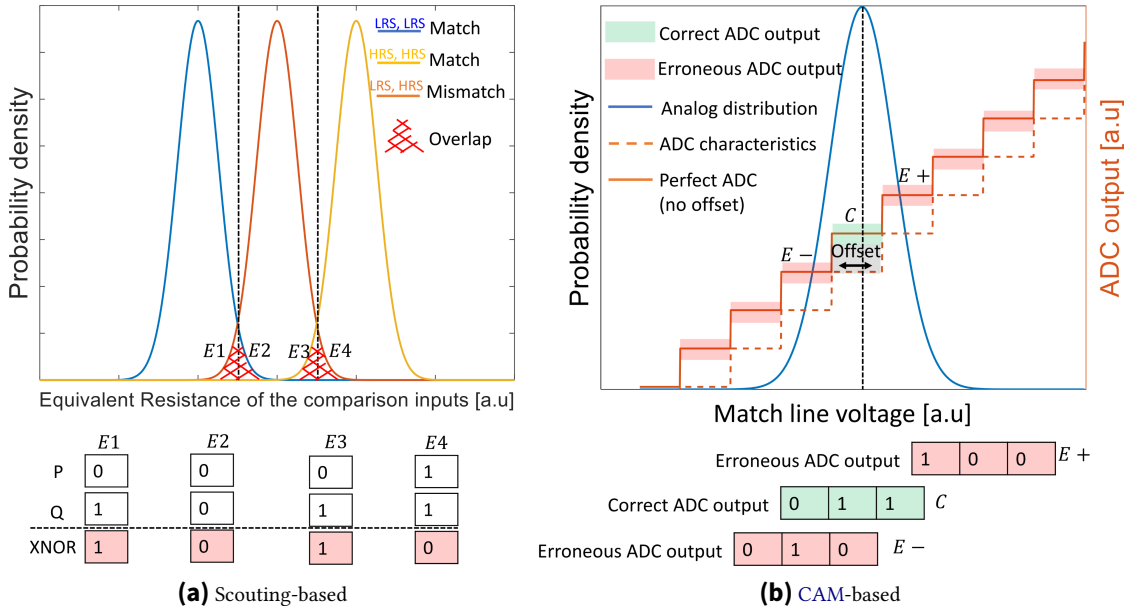

**Figure 5.5.:** Flowgraph of our cross-layer fault modeling, abstraction, and injection methodology, $P_{DF}$ stands for probability of decision failure, $\mu$ and $\sigma$ stands for mean and standard deviation, respectively, also AVF represents the architecture vulnerability factor.

### 5.3.2.2. Circuit-level Error Modeling

As shown in Figure 5.5, due to the different hardware implementations in the scouting-based and CAM-based flavors, we considered different approaches for their circuit-level error modeling.

**Scouting-based flavor:** In this case, we perform a Monte Carlo SPICE simulation on the 1-bit comparator i.e., SA to find the samples of the output currents. By fitting these current samples to a known distribution, estimation of the decision failure probability $P_{DF}$ is performed by numerically calculating the overlap regions shown in Figure 5.6a. Per-bit $P_{DF}$ is a metric used for the architecture-level fault injection in the case of scouting-based flavor.

**CAM-based flavor:** In this case, firstly, we perform the detailed SPICE simulation on the NVM-CAM structure (Figure 5.2) to obtain the equivalent resistance distribution (R)




**(a)** Scouting-based **(b)** CAM-based

**Figure 5.6.:** Visualizing the decision failure in scouting-based and CAM-based similarity computation.

during the NVM-CAM functionality. With these parameters, we build the transient exponential decaying equation for the ML voltage ($V_{\mathrm{ML}}$) given by Equation 5.1. The $\mathrm{V_{DD}}$ is the voltage source and shows the initial voltage of the ML. $t$ is the sampling time, $C$ is the capacitance of the ML, and $R$ is the equivalent resistive discharge path contributed by each individual CAM cell. To account for the process variation, the value of the LRS and HRS consisting of the discharge path is independently sampled from the respective distributions. After this, the ML voltage is fed to an ADC. Due to the variation in the LRS and HRS, the output of the ADC will also undergo certain variations, as shown in Figure 5.6b. The variation of the ADC output is the metric that will be transferred to our architecture level and further be used in our architecture-level fault injection dedicated to this flavor.

$$V_{\mathrm{ML}}(t) = \mathrm{V_{DD}} \left(1 - \exp(\frac{-t}{R \cdot C})\right). \tag{5.1}$$

### 5.3.2.3. Application and Architecture-level Fault Injection

At this level, we performed the Monte Carlo fault injection based on the fault models abstracted from the circuit level, using our architecture-level fault injection handler. For this aim, an identical similarity kernel was executed with multiple ($N_{MC}$) random input sets, and its output was subjected to fault injection based on the obtained hardware-level fault models. As discussed in Section 5.3.2.2, in the case of the scouting-based, the decision failure is injected identically and independently to the output of each XNOR gate. Also for the CAM-based flavor, the modeled variation is injected into the output of the ADC module as a digital value.

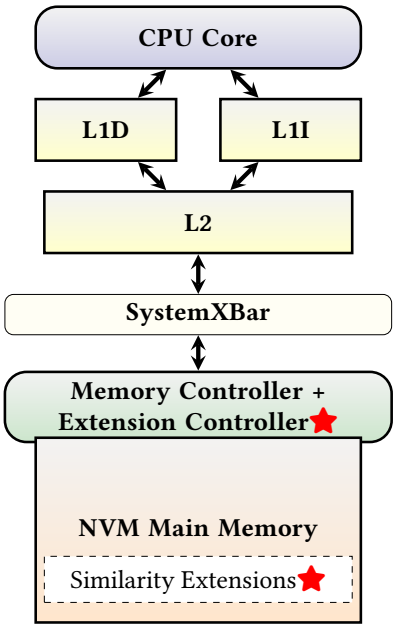



**Figure 5.7.:** Full-system setup hierarchy and the modified hardware components in the conventional structure.

#### 5.3.2.4. Assessing the Architecture and Application-level Masking Capability

Eventually, an important insight that our fault injection can provide, is the error masking capability of the architecture and application. To assess the error masking capability we employed the well-established concept of AVF as the metric [171]. Basically, at the application and architecture-level fault injection, the ratio of the incorrectly calculated application-level results to the total number of executions provided the AVF metric.

## 5.4. Experimental Setup and Evaluation

### 5.4.1. Simulation Setup

In our full-system setup, we utilized existing components from gem5 such as a single X86 Out-of-Order core, two-level cache hierarchy, and main memory. Figure 5.7 demonstrates our Full-system setup hierarchy and Table 5.1 provides detailed information on these setup parameters.

As previously mentioned in Section 5.1, the scouting- and CAM-based similarity computation necessitate different post-operations, influencing overall energy consumption and latency. Accurately modeling both on-chip and off-chip modules in our gem5 simulations posed a challenge. Therefore, we concentrated solely on the energy contribution associated with CiM operations, which encompass factors such as memory to/from CPU read and writes, intra-memory data transfer, and CiM components.

**Table 5.1.:** Main Parameters of the Simulation Setup.

| | |
|---|---|
| NVM models | STT-MRAM [140]<br>ReRAM (JART VCM Read variability)<br>PCM [143, 144] |
| Subarray organization | 256 Rows, 64 columns |
| CPU type | Out-of-Order |
| Compiler & Optimization Flag | g++ -O3 |
| ISA | x86-64 |
| Clock Frequency (GHz) | 1 |
| # of Load/Store Queue Entries | 32 / 32 |
| # of ROB Entries | 128 |
| # of IQ Entries | 64 |
| # of Physical Int/FP/Vec Registers | 128 / 128 / 128 |
| L1 Inst./Data Capacity (KiB) | 32 / 32 |
| L1 Associativity | 2 |
| L1 Tag/Data/Response Latency (ns) | 2 / 2 / 2 |
| L2 Capacity (KiB) | 256 |
| L2 Associativity | 8 |
| L2 Tag/Data/Response Latency (ns) | 20 / 20 / 20 |
| System Bus Latency (ns) | 10 |
| Average Memory Controller Latency (ns) | 15 |
| Read Latency (STT-MRAM/ReRAM/PCM) (ns) | 1 / 1 / 1 |
| Write Latency (STT-MRAM/ReRAM/PCM) (ns) | 4 / 45 / 40 |

We modify the sensing part of an array-level simulator (specifically NVSim [160]) by incorporating energy and latency data obtained from SPICE simulation. For the CAM-based method, we follow the same process as before: simulating the CAM structure in SPICE and integrating the results into the array-level simulator. However, CAM-based similarity computation involves an ADC, a component that greatly affects energy usage and latency. For the ADC, we choose an 8-bit ADC from [243].

We implemented five CiM-friendly applications that leverage CiM for similarity computation, either as their core functionality or as an accelerator. Additionally, we utilized randomly generated input files. This approach aided in observing the system's behavior under various conditions, facilitating a more comprehensive assessment of how the system reacts to different data-dependent inputs. Due to the unique characteristics of each application, we defined unique metrics for measuring its accuracy and subsequently its AVF. These metrics are explained in detail for each application in the following section.

Furthermore, each application was simulated in Monte Carlo fashion across 250 scenarios, with each scenario utilizing a distinct random seed value for input generation and compris-

**Table 5.2.:** Main Assumptions for the end-to-end framework.

| Abstraction of the hardware-level faults | | | | |
|---|---|---|---|---|
| | | STT-MRAM | ReRAM | PCM |
| Scouting-based | Bit Error Rate for $(0 \odot 0)$ | $4.99 \times 10^{-8}$ | 0 | 0 |
| | Bit Error Rate for $(1 \odot 0)$ | $2.94 \times 10^{-5}$ | $5.82 \times 10^{-101}$ | $2.09 \times 10^{-14}$ |
| | Bit Error Rate for $(1 \odot 1)$ | $2.58 \times 10^{-4}$ | $1.01 \times 10^{-97}$ | $2.23 \times 10^{-9}$ |
| CAM-based | HRS mean (Ω) | $3.9478 \times 10^{4}$ | $2.501 \times 10^{5}$ | $2.066 \times 10^{7}$ |
| | HRS std (Ω) | $2.4355 \times 10^{3}$ | $4.495 \times 10^{4}$ | $1.756 \times 10^{6}$ |
| | LRS mean (Ω) | $2.08816 \times 10^{4}$ | $9.862 \times 10^{3}$ | $5.425 \times 10^{4}$ |
| | LRS std (Ω) | $1.2192 \times 10^{3}$ | $5.802 \times 10^{2}$ | $4.732 \times 10^{3}$ |
| | ADC sampling time (ns) | 1 | (for all three NVM technologies) | |
| | Matchline Capacitor (pF) | 2 | (for all three NVM technologies) | |
| Array organization | | | | |
| CAM Capacity per Bank (bit) | | 64 rows × 64-bit + 64-bit Query | | |
| CiM Operation Latency (ns) | | | | |
| Scouting-based | | 448 | 3136 | 2816 |
| CAM-based | | 152 | 488 | 448 |

ing more than $4.25 \times 10^{6}$ bits. Typically, the number of Monte Carlo simulations needs to be sufficient to capture at least one erroneous event. For our purpose, the aforementioned number of Monte Carlo simulations leads to error capturing in all the applications, therefore this number is sufficient in the scope of this work. Increasing the number of Monte Carlo simulations linearly contributes to the total time required for the simulations.

Moreover, for each input pattern, the fault injection methodology invokes the normal distribution generator using the parameters specified in Table 5.2 to determine the cell resistance for CAM-based behavior modeling as discussed in Section 5.3.2.2. For runtime and energy calculation, only a small input size is considered (512 KiB).

On the other hand, the latency difference between different technologies for a specific AM flavor is in the order of milliseconds, although the write latency of ReRAM technology is almost 11× greater than that of STT-MRAM technology. This is due to the full-system simulation environment. In other words, randomly reading 64-bit data from main memory might take more than 100 ns. As a result, when we consider all the parameters for running our CAM applications, there seems to be a negligible difference between these technologies.

### 5.4.2. Hardware-level Fault Abstraction

As discussed in Section 5.3.2, as per the flavor of similarity computation (scouting- or CAM-based) we considered different methodologies for abstract, model, and inject the hardware-level faults. For scouting-based, we passed the per-bit decision failure probability obtained through SPICE simulation as outlined in Table 5.2. For the CAM-based flavor, building the transient exponential decaying equation for the ML voltage was the prerequisite. To account for the process variation for the CAM-based flavor, in each iteration of the architecture-level fault injection, we randomly sampled from the normal distribution with the mean and standard deviation (std) parameters outlined in Table 5.2. Besides, Table 5.2 also the organization and latency parameters associated with scouting-based and CAM-based. Please note that for runtime and energy calculation, only a small input size is considered (512 KiB). These normal distributions, as well as the latency parameters, are the results of our circuit-level SPICE simulations.

### 5.4.3. Application-level Analysis

Our applications are classified into three types based on their characteristics: ❶ Applications with binary output results, such as exact matching and HDC, where no thresholding is employed for output classification. ❷ Applications like DNA sequencing and image hashing, which utilize thresholding for output classification. ❸ Applications like Bloom filters, which incorporate randomness in their tasks without employing output classification thresholds.

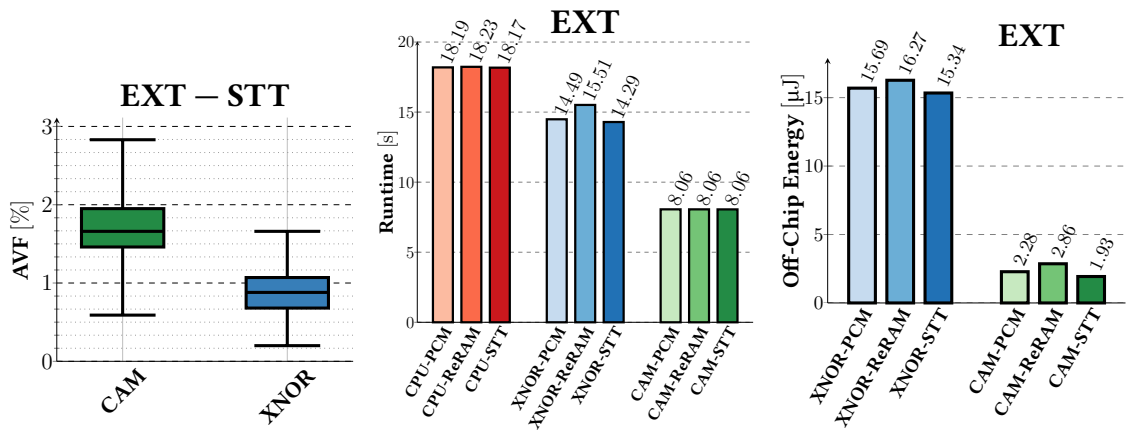
Among these types, Type ❶ applications exhibit the lowest fault tolerance. Type ❷ applications demonstrate moderate fault tolerance. Interestingly, Type ❸ applications, despite their resemblance to Type ❶, exhibit a fault tolerance level similar to Type ❷ and could be the most suitable applications to harness CAM-based flavor capability.

The decision on whether to perform parts of the similarity operation in memory or on the processor is determined by a flag, specifying the similarity computation flavor, passed to the API. With the assistance of the compiler, corresponding assembly instructions for the CPU and operation commands for similarity computation units are generated.

Each application features at least one large main loop, with some containing multiple nested loops to iterate through partial outputs. As a result, the low-level instructions generated vary depending on the application. Furthermore, to ensure fairness in comparing application implementations between these two flavors, as well as with the processor version, all optimization tasks are managed by the compiler.

#### 5.4.3.1. Exact Matching (EXT)

In this application, CAM is utilized to determine the presence of a query in a dataset. Hamming distances are calculated as intermediate results, which are later verified by the




**(a)** AVF percentage **(b)** Runtime **(c)** Energy consumption

**Figure 5.8.:** AVF percentage, Runtime, and Energy consumption of Exact Matching (EXT) application.

CPU. However, the distinction between scouting-based similarity computation and CPU lies in the fact that all XNOR operations between the query and dataset occur within the memory, with the intermediate results subsequently copied to the CPU for final verification.

Here, CPU results serve as the fault-free baseline standard for verifying AM results, and any differences are considered as erroneous. Figure 5.8 displays AVF percentages of these three flavors for STT-MRAM technology. It is noteworthy that the AVF results of the two other technologies are nearly zero, due to the distinguishing gap between their LRS and HRS states. As expected, the AVF of CAM is slightly higher than that of scouting-based XNOR for this simple application.

#### 5.4.3.2. DNA Sequencing (DNA)

Here, the DNA query is compared against all other DNA sequences [244], and if a match or higher similarity (above a predefined threshold) is found, they are considered valid results. Regardless of the similarity flavor, this threshold checking is entirely performed by the CPU.

Similar to the previous application, CPU results are considered as the fault-free baseline. Here, the erroneous results are divided into two subcategories: False Positive (*FP*) (i.e., the DNA sequences that are considered similar by mistake), and False Negative (*FN*) (i.e., the DNA sequences that are considered non-similar by mistake). Figure 5.9 illustrates the distribution of FP and FN results for the DNA sequencing application. The AVF of scouting-based XNOR is near zero, and the large gap between FP and FN results in CAM-based similarity computation is due to the nonlinearity of the ML voltage with respect to the number of matches and mismatches.

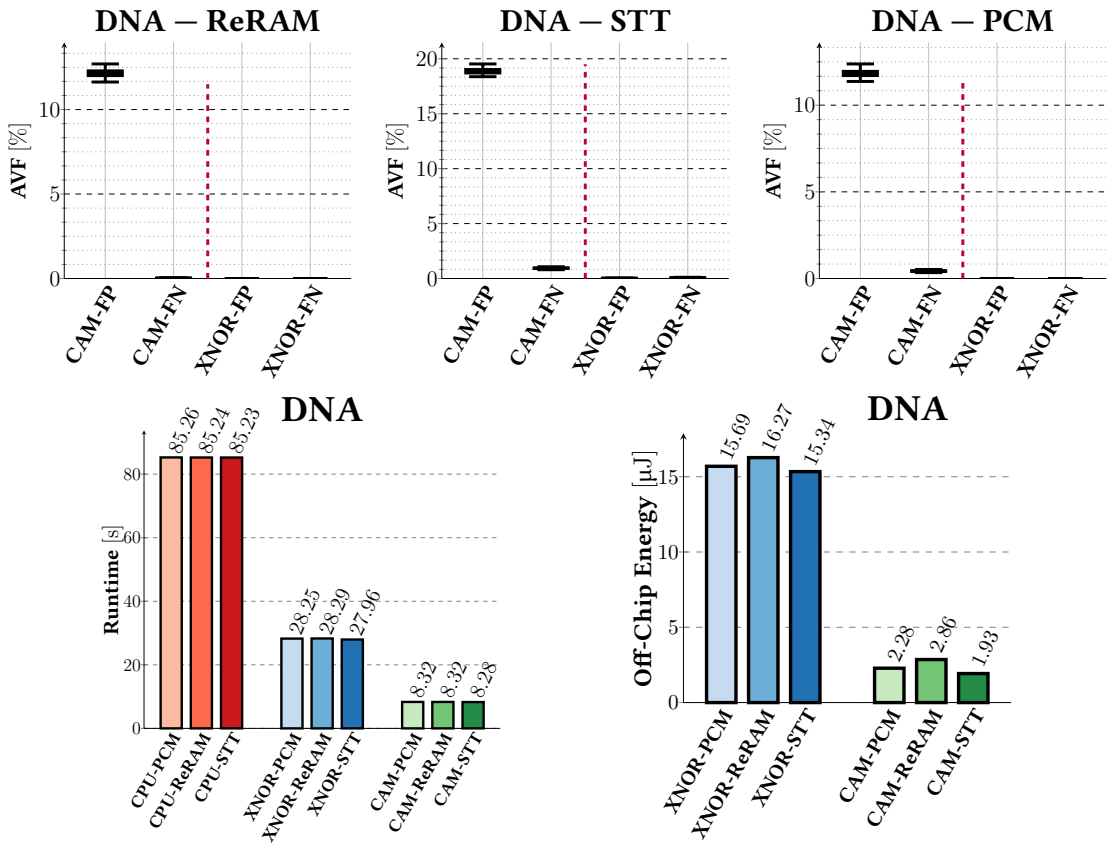



**Figure 5.9.:** AVF percentage, Runtime, and Energy consumption of DNA Sequencing (DNA) application. FP and FN stand for False Positive and False Negative, respectively.

#### 5.4.3.3. Classification (HDC)

Similar to the previous application, hyper-dimensionally encoded vectors [245] are compared with a query vector, and the most similar one is selected as the final answer. However, due to the large size of these vectors, they cannot all fit inside the CAM at once. Therefore, the partial similarity results need to be properly accumulated within the CPU to make the final classification decision.

Also here, any difference compared to CPU results is considered as erroneous. Figure 5.10 demonstrates these results. The large gap between the AVF of CAM-base and scouting-based flavors could be due to the fact that accumulating the partial CAM results together to calculate the total inverse Hamming distance for each class partially accumulates errors, resulting in a large error at the end for CAM results.

#### 5.4.3.4. Image Hashing (IMH)

In this application, we utilize the *Average Image Hashing* method [246] to generate a hash value for large images and employ it for similarity checking. To accomplish this, each block of an image is loaded into the AM. A random query value is then compared with the

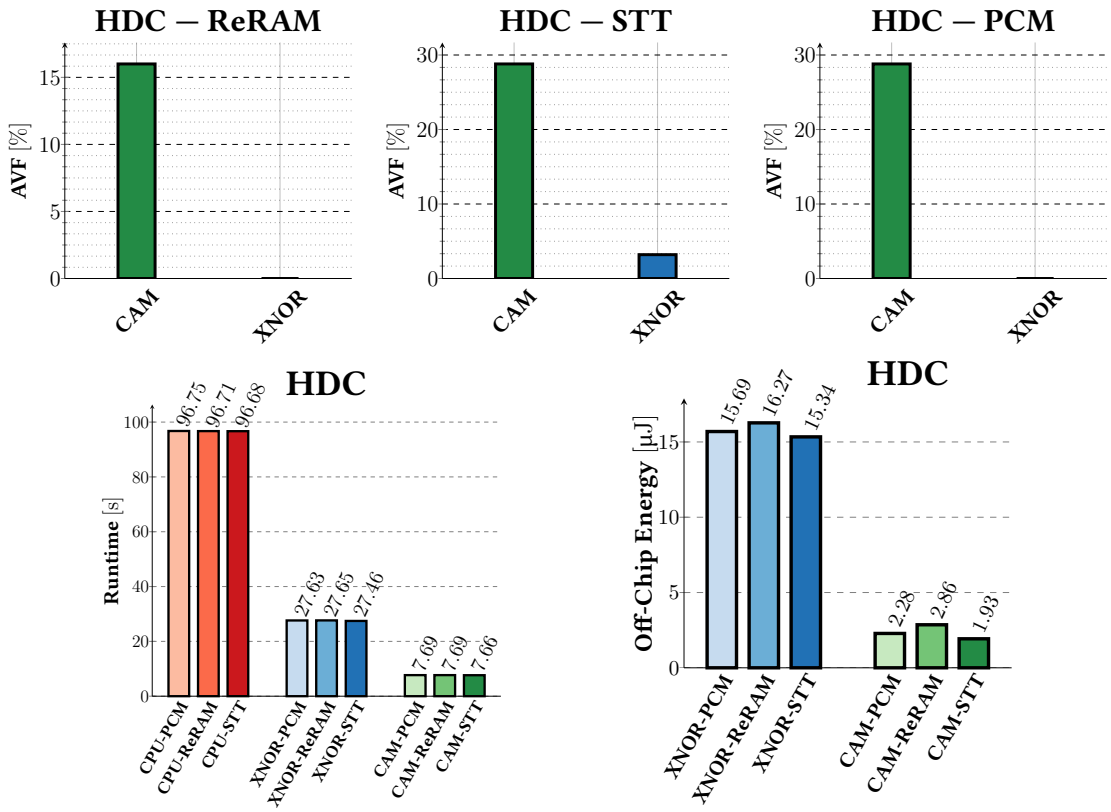



**Figure 5.10.:** AVF percentage, Runtime, and Energy consumption of HDC-based Classification (HDC) application.

data block, and the inverse Hamming distances calculated by the AM or CPU for each row and then accumulated across all the data blocks. Finally, the total accumulation of these partial inverse Hamming distances for each block is reported as the final hash value.

The Average Image Hashing algorithm belongs to the subcategory of approximate algorithms. Even in an fault-free environment like the CPU, correct outputs for similarity checking may not be achieved 100% of the time. Therefore, for our simulation purposes, we insert 5% and 20% random noise into the randomly generated images, respectively. Then, we compare these two images with the original one, and if the difference is below 10%, it is considered similar. Likewise, if it is above 10%, it is considered non-similar. Figure 5.11 shows the FP and FN AVF of the CPU and other NVM-CiM-based similarity computation flavors. Here, the scouting-based flavor leads to the worst AVF results, as both the approximate error of the algorithm and the error of the scouting-based flavor originating from decision failure are added up together in the end.

#### 5.4.3.5. Bloom Filter (BLF)

Here, we utilize the AM as a simple hash function to generate index values for a basic *Bloom filter* [96] with three containers. Each input is loaded into the AM, and for each container, we use different random values for the query. These queries are compared with the input

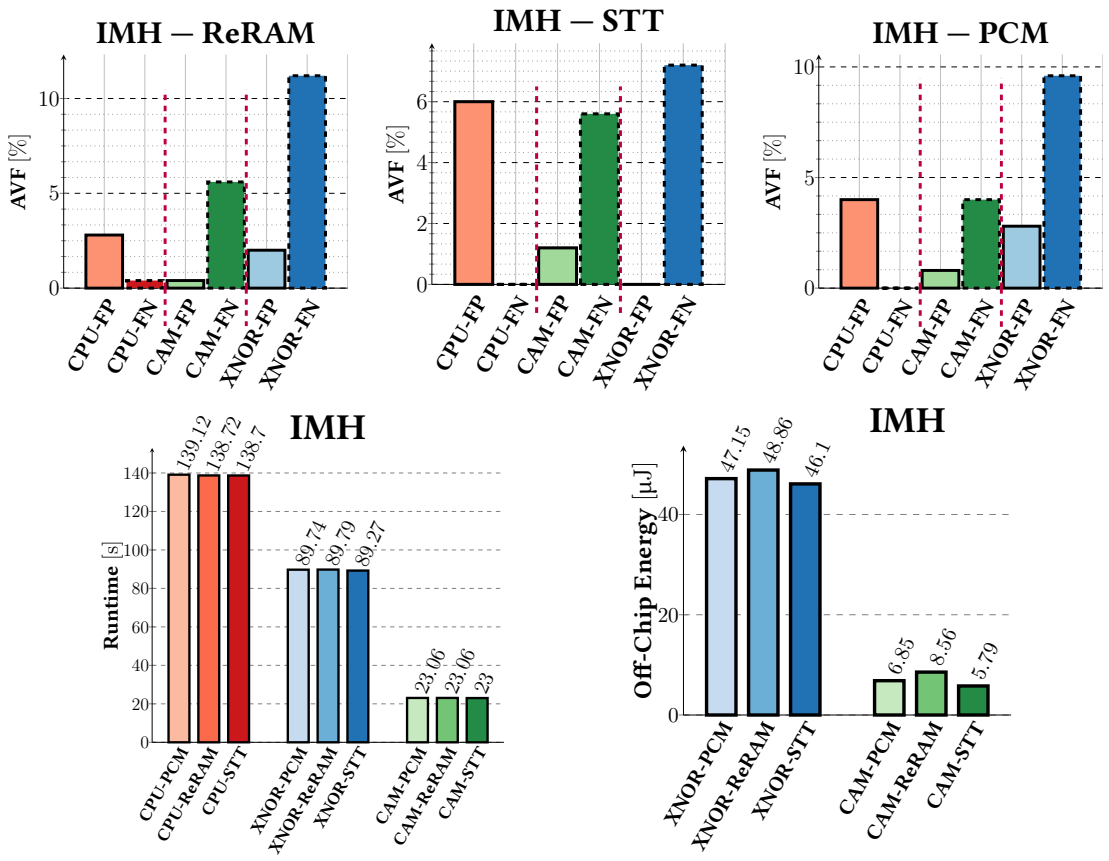



**Figure 5.11.:** AVF percentage, Runtime, and Energy consumption of Image Hashing (IMH) application. FP and FN stand for False Positive and False Negative, respectively.

data, and finally, the accumulated inverse Hamming distances are passed to a basic random function generator to produce a unique index for each container. Again, depending on the NVM-CiM-based similarity flavor, the inverse Hamming distance is either calculated inside the AM or the CPU, but the final accumulation and index generation process are both entirely performed within the CPU.

There should not be any FN error in the concept of a Bloom filter. This means that when something is inserted into the Bloom filter container(s), it should always be detected when checked. However, it is possible to have a FP error, where data is not inserted into the container(s) but is incorrectly detected as inserted.

For our simulation purposes, we generated two categories of data and filled more than 70% of the containers with one of them. Neither the CPU nor any of the NVM-CiM-based similarity computation flavors exhibited FN errors. However, the FP results are shown in Figure 5.12. Interestingly, the results of the flavors are slightly better than those of the CPU in a fault-free environment. This is because the non-ideal noise of the NVM devices helps to introduce more randomness into the index generation process for our Bloom filter containers. As evidence of this, CAM-based flavor, which is a more error-prone environment, performs slightly better compared to CPU and scouting-based flavor in all

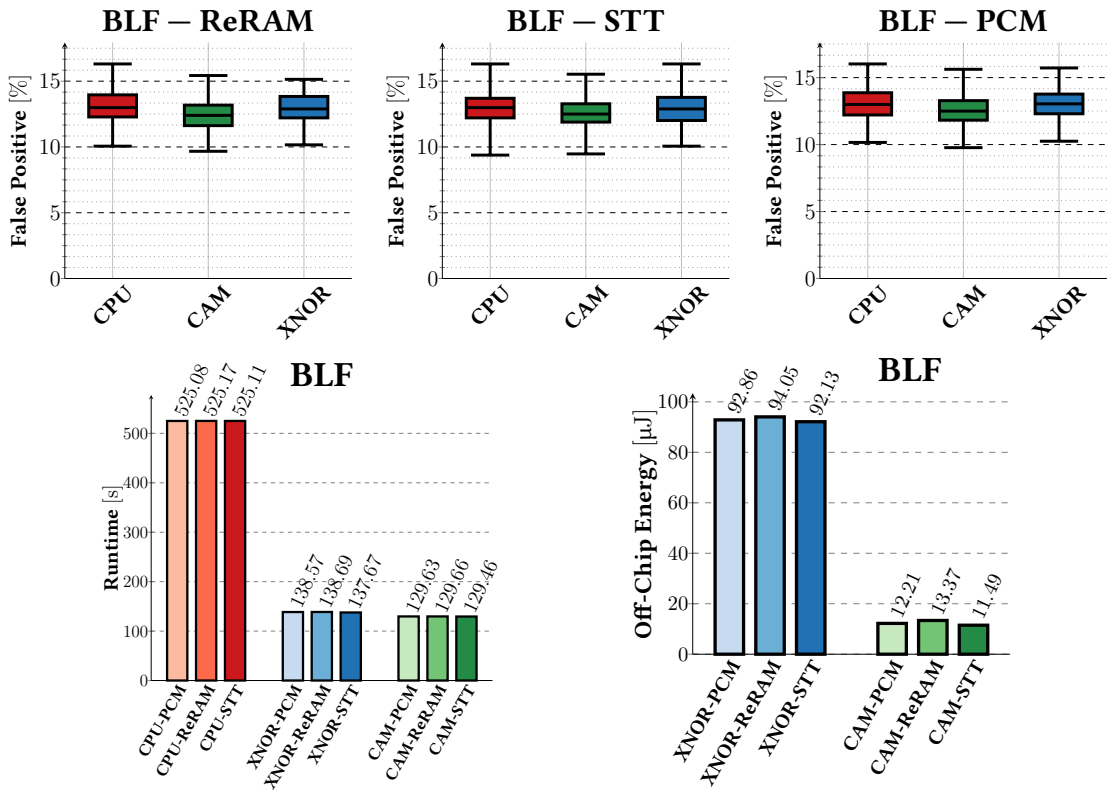



**Figure 5.12.:** False Positive percentage, Runtime, and Energy consumption of Bloom Filter (BLF) application.

three NVM technologies. Overall, this suggests that Bloom filter or similar applications could benefit from AM acceleration.

### 5.4.4. Result Analysis

Regarding the latency, for some applications, there is a small difference between the scouting-based and CPU versions, while for others, this difference is more significant. The reason behind this lies in the scenario where applications have small input sizes compared to the CPU cache capacity. In such cases, prefetching operates effectively, and the program is sufficiently optimized. Thus, whether the scouting-based unit prepares XNORed results or the CPU only loads raw inputs does not matter much, as the CPU can execute compare operations on raw data without requiring scouting units. In contrast, this implies that if more scouting units are integrated into the main memory, enabling parallel processing on larger datasets, the latency difference mentioned earlier could become substantial.

Regarding energy consumption, the CAM-based flavor achieves an average off-chip energy saving of 6×. This reduction is primarily attributed to the significant decrease in data size transferred from memory to the processor, which is reduced by a factor of 8 compared to other factors that remain relatively consistent between the two flavors. Additionally, the utilization of DMA for data loading further supports the observed 6× energy savings, making them expected.

Furthermore, we observe technology-independent behavior regarding both latency and energy consumption across various application types, despite a more than 10× difference in write latency between ReRAM and STT-MRAM technologies. This behavior persists due to the full-system nature of the simulation, which considers all interactions rather than isolated scenarios. Conversely, reliability, regardless of application type, proves to be both technology- and flavor-dependent in most scenarios.

## 5.5. Conclusion

In this chapter, we have characterized in-memory similarity computation using NVM-based CiM architectures and demonstrated its potential to enhance performance by reducing overall energy consumption and runtime latency. We have also emphasized the importance of reliability assessment for representative NVM technologies employed for similarity computation. By comparing different flavors of NVM-CiM-based similarity computation and a diverse set of applications, we have quantified how workload characteristics interact with the underlying hardware.

We have shown that latency and energy are largely technology-independent in our full-system setting, whereas reliability strongly depends on both the memory technology and the similarity-computation flavor, irrespective of application type. Interestingly, some applications, such as Bloom filters and image hashing, can even benefit from the intrinsic non-idealities of NVM technologies. Overall, the scouting-based flavor provides the most favorable reliability profile compared to the CAM-based flavor.

Building on these insights, the next chapter focuses on cross-layer techniques to guarantee reliable *exact-match* operation in NVM-CAM structures. There, we develop an algorithm–technology co-optimization methodology that mitigates sensing errors without sacrificing the performance and energy advantages of large-scale in-memory search.

# 6. Cross-layer Solution for Reliable NVM-CAM Realization

**Literature**
This chapter is based on the following authored publication:
"Algorithm–Technology Co-Optimization for Reliable NVM-CAM Systems"

In Section 2.4, we introduced the basic organization and sensing principles of NVM-CAMs, and in Section 2.6 we highlighted their main reliability challenges. Building on this background, Chapter 5 evaluated NVM-based similarity-computation kernels implemented with both scouting- and CAM-style CiM architectures, primarily targeting approximate search and quantifying the associated reliability, energy, and performance trade-offs. However, these chapters did not address how to guarantee reliable operation or how to achieve high-accuracy results in applications that require exact matching rather than approximate CAM outputs.

To address these limitations, this chapter presents a cross-layer algorithm–technology co-optimization solution to mitigate SeER and improve NVM-CAM reliability without compromising performance. At the circuit level, the matching voltage is tuned to a statistically derived optimal value determined by both technology- and architecture-level parameters. Under this configuration, the proposed NVM-CAMs reliably capture all true matches, with only a small number of unintended false positives. Consequently, at the algorithm layer, the matching procedure is adjusted so that the processor verifies candidate matches and identifies the correct results. This reduces the SeER to 0% with negligible verification overhead, as iterating over a limited subset of candidates is significantly more efficient than traversing the entire data array.

Experimental results show that NVM-CAMs effectively reduce processor–memory data transfers. This alleviates the memory wall and improves both performance and energy efficiency. Compared to an ideal NVM-CAM, the proposed solution effectively addresses matching reliability challenges while incurring only 6% run-time and 4% energy overheads, without any additional hardware cost, thereby complementing the benchmarking and analysis performed in the preceding chapters.

**Table 6.1.:** Representative NVM-CAM implementations.

| Paper | Technology | Cell Configuration | SeER |
|---|---|---|---|
| [107] | STT-MRAM | 3T2R | $1 \times 10^{-4}$ |
| [108] | STT-MRAM | 3T2R | $3.3 \times 10^{-4}$ |
| [100] | STT-MRAM | 10T4R | $1.5 \times 10^{-2}$ |
| [99] | STT-MRAM | 10T2R | $5 \times 10^{-3}$ |
| [103] | STT-MRAM | 12T2R | $1.1 \times 10^{-4}$ |
| [104] | STT-MRAM | 15T4R | $2.7 \times 10^{-2}$ |
| [105] | STT-MRAM | 15T4R | $4.5 \times 10^{-2}$ |
| [106] | STT-MRAM | 15T6R | $5 \times 10^{-3}$ |
| [110] | ReRAM | 2T2R | $3.9 \times 10^{-2}$ |
| [115] | ReRAM | 5T2R | $5 \times 10^{-2}$ |
| [117] | ReRAM | 6T2R | $1 \times 10^{-2}$ |
| [119] | PCM | 2T2R | $1 \times 10^{-4}$ |

## 6.1. Related Work

In the context of reliability-oriented CAM design, numerous NVM-CAM architectures have been proposed based on various NVM technologies, including STT-MRAM [99, 100, 103, 104, 105, 106, 107, 108], ReRAM [110, 115, 117], and PCM [119]. These designs exhibit diverse structural complexities, ranging from compact 2T2R cells [110] to more complex 15T6R cell [106] configurations. Despite continuous architectural improvements, most reported SeERs remain on the order of $10^{-2}$, while the most reliable implementations achieve approximately $10^{-4}$ [107, 119], which is still insufficient for high-reliability applications. Moreover, increasing the number of transistors per cell further exacerbates area overhead, similar to the constraints in SRAM-based CAMs. Table 6.1 summarizes representative NVM-CAM implementations, outlining their cell configurations and corresponding SeER metrics.

## 6.2. Proposed Methodology

This section begins by defining the reliability challenges in NVM-CAMs and explaining how SeER drastically increases in practical NVM-CAM implementations. It then introduces our error-handling methodology to mitigate this problem.

### 6.2.1. Problem Definition

In an NVM-CAM, the ML voltage decay during a search operation can be modeled as an $RC$ circuit, where $C$ is the ML capacitance charged to $V_{dd}$, and $R$ is the equivalent resistance of all activated pull-down paths. Figure 6.1a shows the ML voltage decay for various

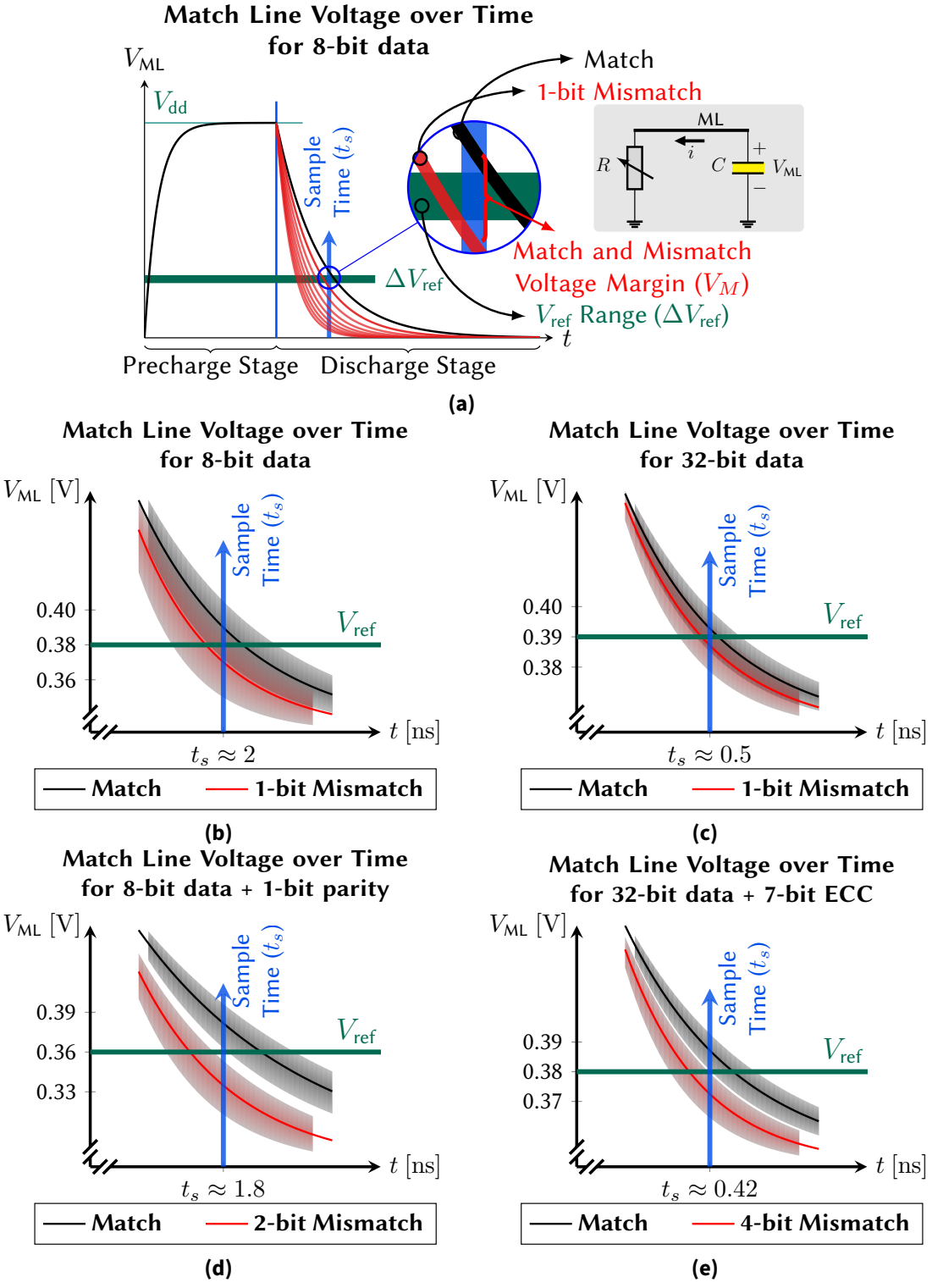



**Figure 6.1.:** Search operation voltage decay for varying numbers of match/mismatch bits in an 8-bit STT-MRAM-based CAM. (a) illustrates the ideal case without process variations, while (b) and (c) show the impact of process variations on the discharge curves for 8-bit and 32-bit data sizes, respectively. (d) and (e) depict the effect of appending parity and ECC bits, which increase the minimum Hamming distance between match and mismatch states. Shaded regions represent voltage distributions resulting from process variations.

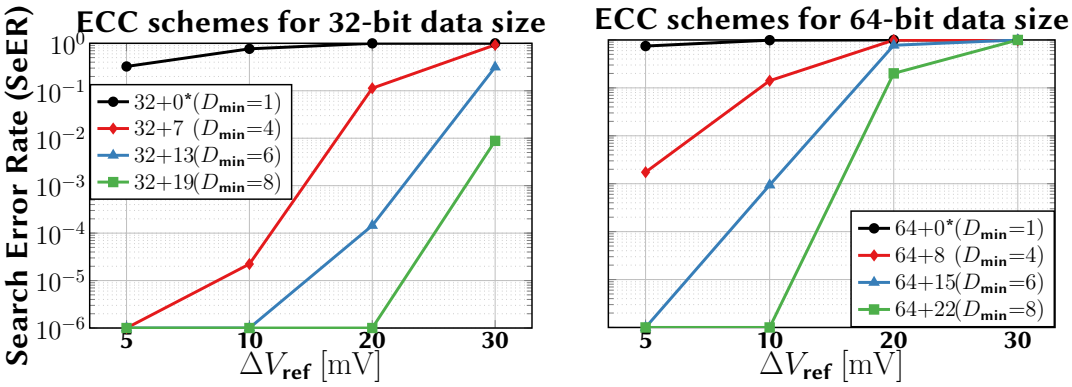



**Figure 6.2.:** SeER analysis of STT-MRAM-based CAM using different ECC schemes for 32-bit and 64-bit data sizes, evaluated across varying $\Delta V_{ref}$ ranges. (* *indicates baseline without ECC*)

match/mismatch combinations in an 8-bit (i.e., column size) STT-MRAM-based CAM under ideal conditions. For an $n$-bit CAM, there exist $n$+1 discharge levels, corresponding to Hamming distances from 0 (full match) to $n$ (full mismatch).

A comparison between Figure 6.1b and Figure 6.1c indicates that increasing $n$ (from 8-bit to 32-bit in this instance) results in a reduction of $R$, thereby accelerating the discharge process and enabling faster search operations. However, a higher $n$ also increases the number of discharge levels and reduces the voltage margin ($V_M$) between adjacent states. Furthermore, process variations induce a probabilistic distribution in each discharge path (represented by the shaded regions in Fig. 6.1), which can further diminish or even cause overlap of $V_M$. Consequently, the reduction in $V_M$ impairs the ability to distinguish between match and mismatch cases, thereby leading to an increase in the SeER.

In practice, $n$ must be sufficiently large (e.g., 64-bit or higher); however, increasing $n$ exacerbates the reduction in $V_M$ and, consequently, increases in SeER. A well-established approach to improve $V_M$ while maintaining a large $n$ is to increase the minimum Hamming distance ($D_{\text{min}}$) between match and mismatch states through parity or Error-Correcting Code (ECC) (ECC) bits, to easily distinguish these states. As illustrated in Figure 6.1d–Figure 6.1e, this approach requires no structural modifications or decoding circuitry and can raise $D_{\text{min}}$ to 2 when using parity bits or to 4 when using Single Error Correction, Double Error Detection (SECDED) ECC. Nonetheless, it incurs additional memory overhead and increased power consumption.

Despite the benefits of ECC, its effectiveness diminishes when SA variations are taken into account by treating $V_{\text{ref}}$ as a range ($\Delta V_{\text{ref}}$) rather than as a fixed value. Figure 6.2 demonstrates that the SeER increases sharply with larger $\Delta V_{\text{ref}}$. For example, in a 64-bit STT-MRAM-based CAM, maintaining SeER $<10^{-6}$ with $\Delta V_{\text{ref}}\approx$10 mV requires 22 redundancy bits ($D_{\text{min}}$=8). Nevertheless, this level of redundancy and the relatively small $\Delta V_{\text{ref}}\approx$10 mV remain insufficient for practical NVM-CAM implementations. These limitations indicate that conventional circuit-level approaches alone cannot ensure reliable search operations under ML and SA variations.

**Table 6.2.:** Comparison of $V_M$ and SeER under ideal and non-ideal conditions for STT-MRAM-based CAMs using baseline, parity, and ECC schemes. The best-case $V_M$ and worst-case SeER are highlighted for clarity and comparison.

| Approach | (Data+Redundancy) bits per Row | $D_{\min}$ | $V_M$ [mV] | SeER @$\Delta V_{\text{ref}}$=5 mV |
|---|---|---|---|---|
| Baseline | (64 + 0) | 1 | 4 | $7.461\times10^{-1}$ |
| | (32 + 0) | 1 | 8 | $3.251\times10^{-1}$ |
| | (16 + 0) | 1 | 16 | $4.659\times10^{-2}$ |
| | (8 + 0) | 1 | 31 | $1.303\times10^{-3}$ |
| **Parity** | (64 + 1) | 2 | 8 | $2.707\times10^{-1}$ |
| | (32 + 1) | 2 | 15 | $1.080\times10^{-2}$ |
| | (16 + 1) | 2 | 29 | $2.100\times10^{-5}$ |
| | (8 + 1) | 2 | 53 | $< 10^{-6}$ |
| **ECC** | (64 + 8) | 4 | 14 | $1.738\times10^{-3}$ |
| | (64 + 15) | 6 | 19 | $< 10^{-6}$ |
| | (64 + 22) | 8 | 23 | $< 10^{-6}$ |
| | (32 + 7) | 4 | 25 | $< 10^{-6}$ |
| | (32 + 13) | 6 | 33 | $< 10^{-6}$ |
| | (32 + 19) | 8 | 38 | $< 10^{-6}$ |

Similarly, Table 6.2 compares the $V_M$ between perfect match and $D_{\min}$-bit mismatch scenarios under ideal conditions (excluding non-idealities and process variation) for baseline, parity, and ECC-based schemes. It also reports the resulting SeER at $\Delta V_{\text{ref}}$=5 mV, incorporating the effects of non-idealities and variation. To aid comparison, best-case $V_M$ values and worst-case SeER are highlighted. As seen in Table 6.2, most $V_M$ values fall below 30 mV, implying that differentiating match from mismatch becomes infeasible at higher $\Delta V_{\text{ref}}$. Consequently, the SeER approaches 1 (~100%), as also shown in Figure 6.2.

In summary, as $n$ increases and ML/SA variations grow, the voltage distributions associated with different Hamming distances increasingly overlap, causing SeER to rise sharply. Moreover, redundancy-based techniques (parity/ECC) cannot maintain sufficient voltage margin under realistic $\Delta V_{\text{ref}}$. Therefore, the problem addressed in this work is to ensure reliable NVM-CAM search operations under substantial ML and SA variations while avoiding excessive redundancy or structural changes to the NVM-CAM array. Specifically, we aim to develop an algorithm–technology co-optimization method that mitigates SeER in large-scale NVM-CAMs to support the reliable and efficient acceleration of CAM-friendly, data-intensive workloads.

## 6.3. Proposed Error Mitigation Methodology

To address NVM-CAM reliability problem and effectively mitigate SeER, we propose an algorithm–technology error mitigation methodology. This approach combines circuit-level tuning with software-level filtering to guarantee reliable search in large-scale NVM-CAMs. The core principle is to ensure detection of all true matches (true positives), even if this occasionally produces false positives due to circuit variabilities. Any resulting false positives can then be efficiently filtered in software with minimal overhead and ensuring **zero** SeER. The methodology consists of two key steps:

1. **Circuit-level characterization and tuning:** Analyze the statistical distributions of HRS and LRS resistances in the target NVM technology to determine an optimal $V_{\text{ref}}$ that guarantees all true matches are captured, even under worst-case device variations, while allowing a limited number of false positives. The default $V_{\text{ref}}$ is then lowered to this fixed, optimized level, enabling deterministic hardware operation and reliable capture of all valid matches with no latency or energy overhead, and at the cost of an increased number of candidate matches.
2. **Algorithmic filtering:** At the hardware level, the tuned $V_{\text{ref}}$ guarantees "no false negatives", meaning every true match is always included among the retrieved candidates, even though this may increase the number of false positives. A lightweight software routine then processes these "potential" matches, performing a precise check on the processor to distinguish true matches from hardware-generated false positives. This post-verification step ensures zero SeER while keeping the filtering overhead minimal.

### 6.3.1. Circuit-level Characterization and Tuning

Our approach starts with determining an optimal $V_{\text{ref}}$ that maximizes detection reliability. Device variations affect HRS and LRS states, altering ML discharge rates and potentially causing valid matches to fall below $V_{\text{ref}}$ at $t_s$, resulting incorrect mismatch. To prevent this, $V_{\text{ref}}$ must be set according to the worst-case discharge behavior of valid match cases. Instead of placing $V_{\text{ref}}$ midway between the nominal match and 1-bit mismatch voltages, we set it to correspond to a $k$-bit mismatch. This ensures all true matches are detected while tolerating limited false positives.

The effective ML resistances for a full match ($\mathbf{R}_n$) and a $k$-bit mismatch ($\mathbf{R}_{n-k}$) in an $n$-bit row, considering variability in the resistances of memory cells, can be expressed in terms of the individual cell resistances. Where $\mathcal{N}(\mu_H, \sigma_H^2)$ and $\mathcal{N}(\mu_L, \sigma_L^2)$ represent the distributions

**Table 6.3.:** Minimum $k$ values for STT-MRAM and ReRAM-based cells.

| $n$ | + ECC Bits | $k$-STT-MRAM | $k$-ReRAM |
|---|---|---|---|
| 32 | | 4 | 1 |
| 32 | +7 | 4 | 1 |
| 64 | | 5 | 2 |
| 64 | +8 | 5 | 2 |
| 128 | | 7 | 2 |
| 128 | +9 | 7 | 2 |
| 256 | | 9 | 2 |
| 256 | +10 | 10 | 2 |

of resistance for each cell in the HRS and LRS states, respectively. Then, the effective resistances are given by Equation 6.1 and Equation 6.2.

$$\frac{1}{\mathbf{R}_n} = \sum_{i=1}^{n} \frac{1}{\mathcal{N}(\mu_H, \sigma_H^2)} \tag{6.1}$$

$$\frac{1}{\mathbf{R}_{n-k}} = \sum_{i=1}^{n-k} \frac{1}{\mathcal{N}(\mu_H, \sigma_H^2)} + \sum_{i=1}^{k} \frac{1}{\mathcal{N}(\mu_L, \sigma_L^2)} \tag{6.2}$$

We then solve for $k = \min\{k \mid \mathbf{R}_n > \mathbf{R}_{n-k}\}$, identifying the smallest $k$ that eliminates incorrect mismatches while minimizing false positives. Monte Carlo simulations with $10^8$ samples per pull-down path statistically evaluate these resistance distributions, assuming Independent and Identically Distributed (IID) cells [136].

Table 6.3 lists the minimum $k$ for STT-MRAM- and ReRAM-based 2T2R cells, with and without ECC bits for additional false positive reduction. For instance, in a 32-bit array, STT-MRAM-based CAMs must tolerate up to 4-bit mismatches, whereas ReRAM-based CAMs tolerate only 1-bit mismatches. This highlights that, even for ReRAM with a higher HRS-to-LRS ratio, $V_{\text{ref}}$ should be set to allow a 1-bit mismatch, rather than positioned at the midpoint between the match and 1-bit mismatch, to ensure reliable functionality. As $n$ increases, larger $k$ values are needed for robustness. Overly conservative tuning of $V_{\text{ref}}$ further improves reliability at the expense of energy and performance.

Based on the provided minimum $k$ values (Table 6.3) and technology parameters (e.g., $n$, $V_{\text{dd}}$, $C$, $t_s$, etc.), the corresponding optimal $V_{\text{ref}}$ is determined. In other words, optimal $V_{\text{ref}}$ is equal to $k$-bit mismatch discharge voltage at $t_s$. The original $V_{\text{ref}}$ is then set to this optimal value without any additional hardware modifications or overhead. It is important to note that, in our approach, $V_{\text{ref}}$ is fixed during design time and does not need to be adjustable at runtime.

### 6.3.2. Algorithmic Filtering

Following the circuit-level tuning described above, the search algorithm is modified to operate with NVM-CAM outputs that represent potential matches rather than guaranteed matches. The tuned $V_{\text{ref}}$ ensures that all true matches are always included among the candidates, but it may also allow a limited number of false positives. The software-side filtering step is therefore responsible for refining this candidate set.

In the baseline processor-only workflow, the processor sequentially scans the data array to identify all true matches. With NVM-CAMs, this initial search is instead performed inside the memory array, where the CAM efficiently produces a result vector that marks all potential matches. This change provides two key advantages. First, executing the search directly within memory reduces processor–memory interactions, since the data remains in place and no longer needs to be streamed through the cache hierarchy. Second, the CAM output is a compact result vector, typically 64× smaller than the underlying dataset and often further compressible, which allows efficient caching and rapid iteration over candidate entries.

Once the processor receives the result vector, it checks each indicated entry against the original query using standard processor instructions. This final verification step removes any false positives introduced by the approximate nature of CAM matching at the tuned $V_{\text{ref}}$. Because the number of candidates is far smaller than the full dataset, the additional computation required for this check is minimal. The combination of hardware-side inclusiveness and software-side precision guarantees that the final results are free of SeER.

Algorithm 1 outlines these required software-level steps for utilizing the NVM-CAM. Lines 3–14 perform a search for the query if the array is already stored in the NVM-CAM. Lines 15–16 execute a standard search when the array size is small. Lines 17–25 check the array size, load it into the NVM-CAM appropriately, and then perform the search operation.

## 6.4. Full-System Behavior Modeling Platform

To comprehensively evaluate NVM-CAMs, our methodology spans the circuit level, system level, and software level, capturing both performance and reliability effects across the full stack. This integrated approach enables accurate end-to-end modeling and validation, beginning with circuit-level behavior, extending through system-level architecture, and ultimately incorporating software-level interactions. Figure 6.3 presents an overview of our full-system modeling and validation platform. The following subsections discuss each level in detail.

**Algorithm 1** Algorithm for Query Search While Utilizing NVM-CAM

**Require:** Array, Query $Q$
**Ensure:** Result Vector `result`

```
a* ← Pointer to first element of Array
b* ← Pointer to last element of Array
if a*, b* ∈ NVM_CAM_LUT then
    CAM_Result_Vector ← NVM_CAM_Search(Q)
    for i = 0 to sizeof( CAM_Result_Vector ) − 1 do
        for j = 0 to sizeof( NVM_CAM_rows ) − 1 do
            if CAM_Result_Vector[i] & (1 ≪ j) then
                index ← i × sizeof( NVM_CAM_rows ) + j
                if Array[index] = Q then
                    result.push({index : Array[index]})
                end if
            end if
        end for
    end for
else if (b* − a*) < 1 MB then
    return std::search(a*, b*, Q)
else
    if (b* − a*) > NVM_CAM_size then
        Divide Array into two arrays
    else
        Copy [a*, b*] into NVM_CAM
        Update NVM_CAM_LUT with (a*, b*)
    end if
    Repeat this algorithm
end if
return result
```

### 6.4.1. Circuit-Level Modeling

To develop an accurate, system-level compatible model of circuit behavior, we first construct a statistical model of a single bit-cell consisting of an STT-MRAM or ReRAM in series with an NMOS transistor. The circuit is implemented in Cadence Virtuoso, and 1,000 Monte Carlo simulations are performed per bit-cell to analyze ML discharge characteristics, representative of search operations. Figure 6.4 showcases resistance swing of a single STT-MRAM bit-cell in both HRS and LRS states under varying ML voltage during the read/search operation. Since the resulting behaviors follow a normal distribution, we extract the mean ($\mu$) and standard deviation ($\sigma$), as illustrated in Figure 6.3.

The extracted $\mu$ and $\sigma$ values for each cell state are then used in our Python analysis to perform $10^8$ Monte Carlo simulations, capturing a wide range of variations from best-case to worst-case scenarios. This allows us to calculate the optimal values for $k$ and $V_{\text{ref}}$, as discussed in Section 6.3.

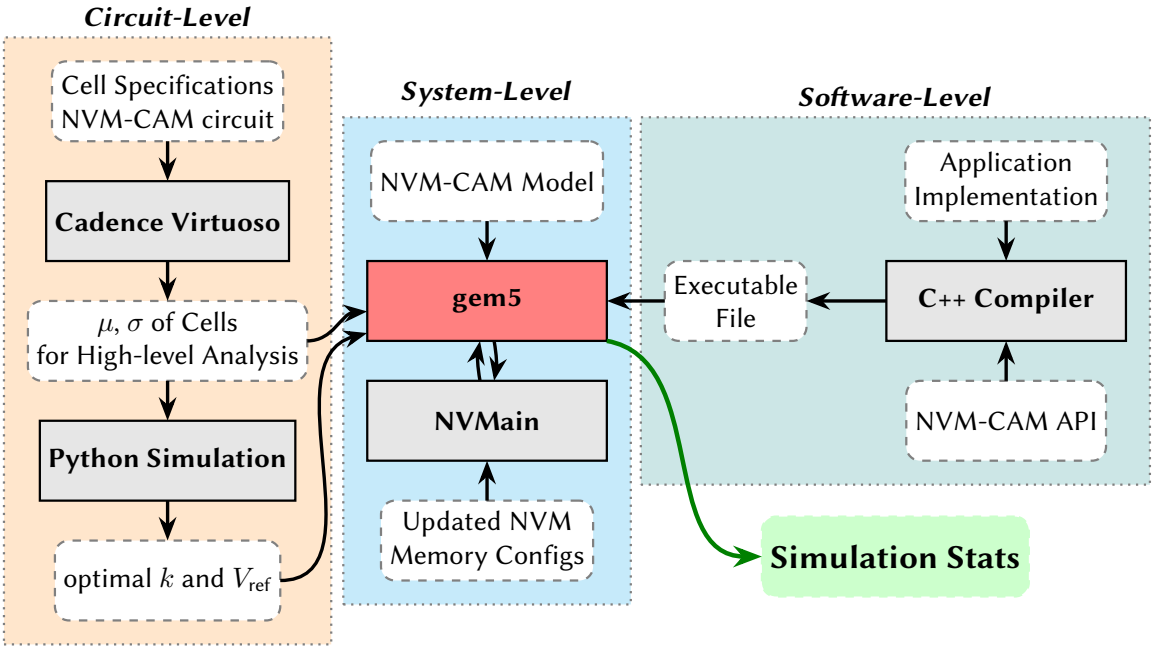



**Figure 6.3.:** High-level overview of the full-system modeling and validation methodology.

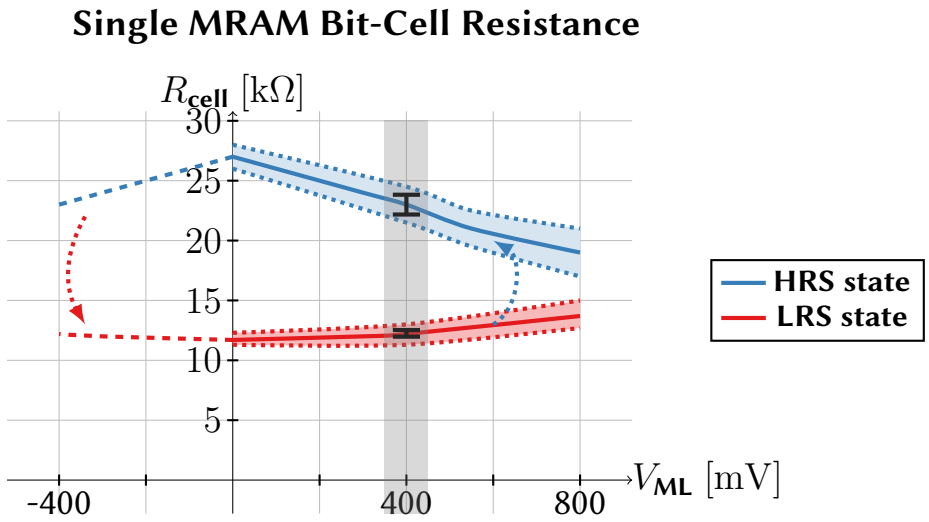



**Figure 6.4.:** Resistance swing of a single STT-MRAM bit-cell in both HRS and LRS states under varying ML voltage during the read/search operation.

In our system-level simulations, each bit-cell is modeled stochastically, capturing its analog behavior by sampling from the corresponding normal distribution every time the cell is accessed. This approach not only represents resistance variations and reliability effects but also implicitly accounts for the bit-cell's relative position to the SA, temperature variations, and SA inaccuracies over time. Therefore, both the extracted $\mu$ and $\sigma$, along with $k$ and $V_{ref}$, are incorporated into our system-level simulations, as shown in Figure 6.3.

### 6.4.2. System-Level Modeling

To model system-level behavior, we employed gem5, a widely recognized full-system simulation framework. By default, gem5 provides a comprehensive simulation environment encompassing a wide range of system components, from CPU cores to the memory subsystem. Since our focus is on in-memory NVM-CAMs, we replaced gem5's default memory controller and main memory model with NVMain [222], which enables detailed simulation of main memory behavior, particularly for NVM technologies. Although NVMain does not natively support in-memory NVM-CAMs, their distinctive behavior is primarily confined to the array level, leaving other characteristics largely unchanged.

As shown in Figure 6.3, to enable accurate simulation of in-memory NVM-CAMs, we extend both gem5 and NVMain. In gem5, we develop a parametric NVM-CAM model that includes default specifications for STT-MRAM and ReRAM. Users can modify key parameters, such as resistance, cell size, and $V_{\text{ref}}$, to study performance trade-offs or to adapt the model to a new memory technology. Additionally, we incorporate multiple reliability models, including CPU-only and ideal, process-variation-aware (our proposed method), and ECC-augmented NVM-CAM configurations. Additionally, NVMain is updated with the latest configurations for commercial NVM technologies. We also add NVM-CAM-specific latency and energy penalties based on worst-case estimates from prior tools [80, 247, 248].

Figure 6.5 shows a simplified system-level configuration of our in-memory NVM-CAM. By default, *Channel#2* is disconnected, and there is no NVM-CAM memory in the CPU-only configuration. We utilize *Channel#2* to separate the physical address spaces of NVM-CAM and DRAM-based main memory. As shown in Figure 6.5, it consists of a single NVM-CAM chip (i.e., no data interleaving), comprising a CAM controller and multiple NVM-CAM subarrays. The CAM controller is similar to a DRAM chip controller, with the key difference being its use of a distinct state machine and dedicated circuitry to orchestrate CAM write operations, retrieve results (rather than subarray content), and update the query. ECC decoding is also performed at this level for ECC-augmented CAM scenarios. Additionally, Figure 6.5 illustrates the communication lines, which include the data I/O bus, row-address lines, and control/select signals. The control/select signals activate a specific subarray, manage CAM-specific operations, and determine the source or destination buffer of the data I/O bus.

### 6.4.3. Software-Level Interactions

As illustrated in Figure 6.3, gem5 requires a software-level executable to serve as the entry point for system-level simulation. Communication between the software and the NVM-CAM is facilitated through the NVM-CAM API. This process is fully automated at compile time by wrapping standard search functions with API calls, as discussed in Section 6.3.2, which provides the actual implementations. As depicted in Figure 6.6, standard CPU-based search functions are replaced with calls to the NVM-CAM API. The API orchestrates interactions between the CPU and the NVM-CAM, ultimately returning

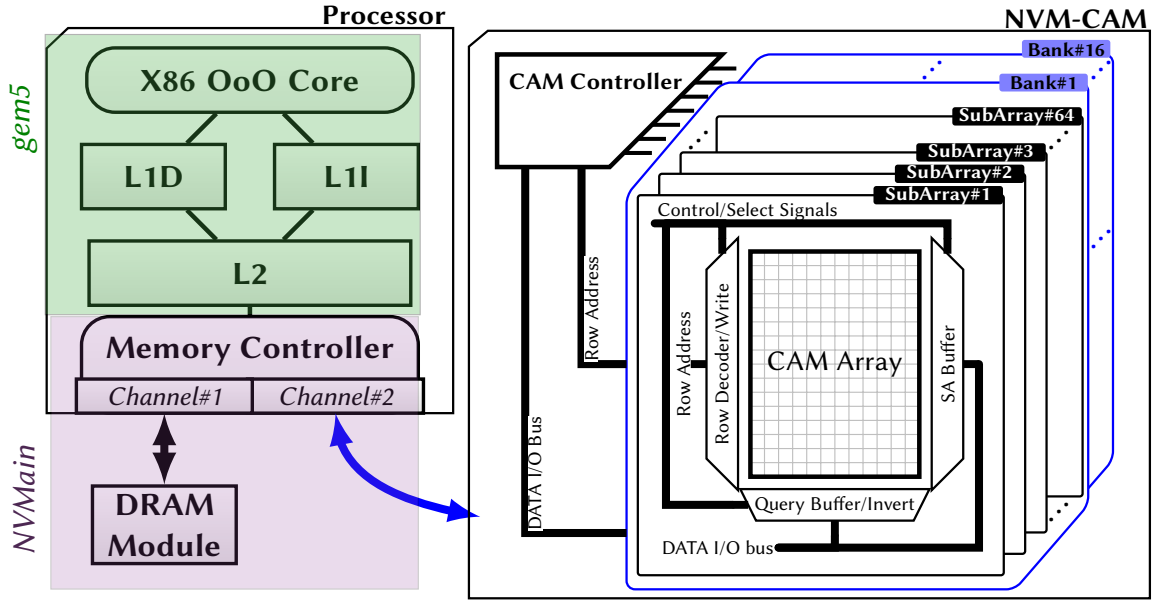



**Figure 6.5.:** System-level configuration of our in-memory NVM-CAM. *Channel#2* is used to map the NVM-CAM into a separate physical address space, isolated from DRAM.

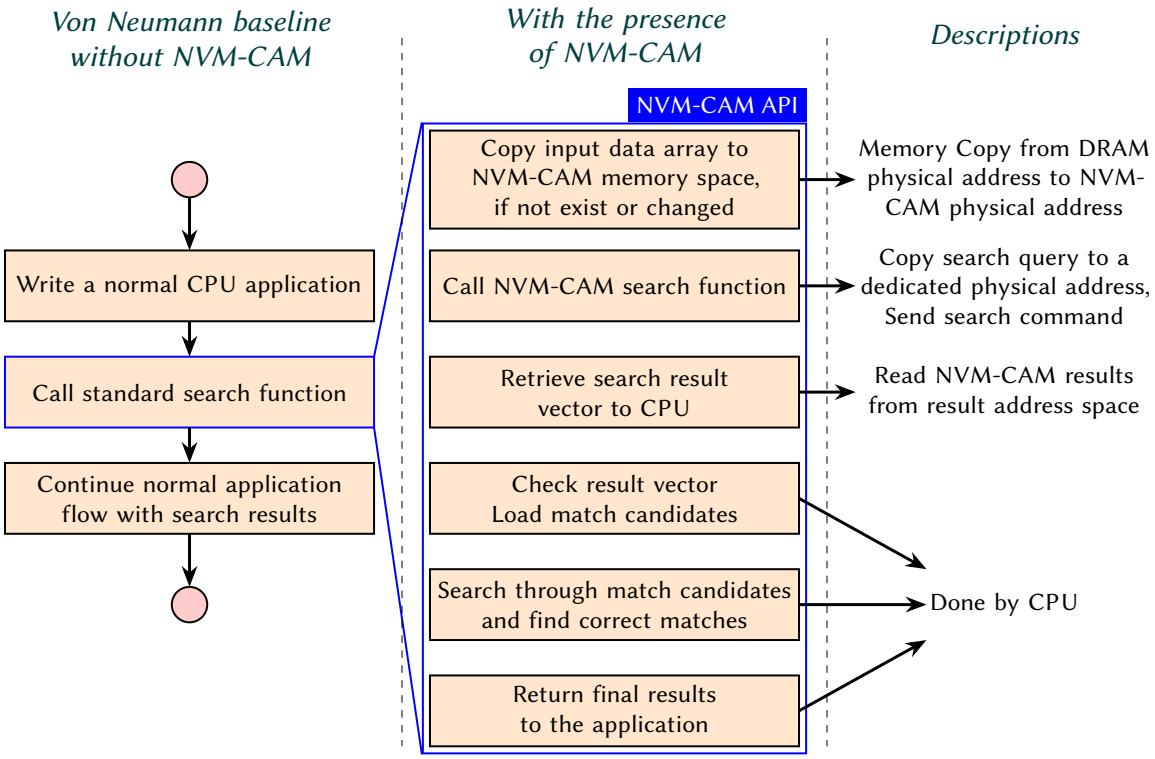



**Figure 6.6.:** Required software-level modifications to utilize NVM-CAM hardware

search results in the required format. Finally, the CPU performs additional verification to ensure the correctness of the NVM-CAM output and guarantees zero SeER before reporting the results.

## 6.5. Evaluation and Results

We evaluated the proposed method across multiple applications, system configurations, and two NVM-CAM technologies (STT-MRAM, ReRAM), using three metrics: *runtime* (total execution time), *off-chip energy* (main memory + NVM-CAM), and *False Positive (FP) ratio* (total matches to true matches; 1 = ideal). While reliability is perfect, FP ratio captures false-positive overhead.

We simulated four different system configurations:

1. *CPU* (Baseline): Uses a conventional CPU–memory system without any NVM-CAM, serving as a reference point for performance and energy comparisons.
2. *IdlCAM* (Ideal CAM): Incorporates an ideal NVM-CAM where all accesses are assumed perfect—free from non-idealities stemming from process variations, noise, or randomness—providing an upper bound on achievable performance and a reference for over-conservative $k$-values than those reported in Table 6.3, which lists the minimum required values.
3. *ProposedCAM* (Realistic analog/stochastic CAM): Introduces realistic analog behavior and process variation; each accessed bit undergoes stochastic modeling using a corresponding normal distribution to emulate the resistance characteristics of NVM-CAM cells, thereby capturing the impact of physical non-idealities on system behavior.
4. *EccCAM* (ECC-augmented CAM): Extends *ProposedCAM* by appending ECC bits to each data entry. These bits increase $D_{\text{min}}$ between matching and mismatching cases, thereby helping to reduce the the likelihood of false positives.

In *EccCAM* configuration, we use standard SECDED coding, which provides $D_{\text{min}}$ of 4, and include the associated energy consumption of the ECC decoding. However, it is important to note that *EccCAM* is incompatible with ternary/masked queries, as standard ECC coding schemes do not support "don't care" values. Therefore, for one of our applications—*DNA*, which relies on ternary/masked queries—we do not report results under this configuration.

To investigate the impact of dataset size on performance and reliability, we tested two dataset classes. ❶ Small datasets (approximately 10 MB) represent lightweight workloads where NVM-CAM can accommodate most or all data without frequent replacements. In contrast, ❷ large datasets (approximately 100 MB) emulate data-intensive applications that frequently overwrite NVM-CAM content due to limited capacity.

To capture realistic system behavior, our simulation framework integrates per-bit stochastic modeling and analog behavior emulation, which significantly increases computational complexity. To maintain tractable simulation times while preserving realistic behavior, we adopted a two-level cache hierarchy with an L2 cache of 2 MB and varied NVM-CAM sizes ranging from 8 MB to 32 MB, depending on the column size of the NVM-CAM. This setup enables scaled-down simulations that still reflect realistic cache–memory interactions.

While small datasets tend to remain stationary in NVM-CAM, large datasets cause frequent content updates, highlighting the limitations of finite NVM-CAM capacity. This variability enables the exploration of both steady-state operation and high-load scenarios. Although the simulation incurs significant overhead, execution times remain manageable—ranging from several hours to two days—while providing comprehensive performance insights across diverse scenarios.

We validated our approach using three real-world applications, each representing a distinct workload domain:

1. *DB* (Database Search): Involves saving hashed table records into NVM-CAM for rapid retrieval. We used the Steam video game recommendation dataset [249] as the small dataset, and the myKet Android application installation dataset [250] as the large sample datasets.
2. *ISH* (Image Similarity Hashing): Applies Average Hashing [246] to generate multiple hash signatures per image. These partial hashes are stored in NVM-CAM to facilitate efficient similarity queries, using the CIFAR-10 dataset [251].
3. *DNA* (DNA Sequence Search): Searches for specific nucleotide sequences within a real human chromosome dataset [252]. Since biological sequences may not align to word or byte boundaries, this task requires support for bit-level shifting and approximate matching—making it a strong candidate for evaluating variability-tolerant CAM architectures.

Table 6.4 summarizes all the parameters used in the circuit- and system-level simulation setup, along with the corresponding applications and datasets. Similarity, Figure 6.7, Figure 6.8, and Figure 6.9 presents the runtime, off-chip energy, and FP ratio, respectively, which are discussed in the following. Although each scenario emphasizes different behaviors depending on the nature of the application, we focus on the geometric mean results for a fair comparison and discuss them accordingly.

**Runtime:** ReRAM-*IdlCAM* exhibits approximately 0.1% higher latency than STT-MRAM-*IdlCAM*, primarily due to device-level ReRAM configuration characteristics. Nevertheless, because ReRAM offers a more favorable FP ratio than STT-MRAM, ReRAM-*ProposedCAM* incurs only about a 3.4% runtime overhead, whereas STT-MRAM-*ProposedCAM* introduces roughly a 6.7% overhead. Employing *EccCAM* reduces runtime by approximately 2% for STT-MRAM; however, for ReRAM, the additional ECC overhead results in a 0.7% increase in runtime. Overall, compared with conventional von Neumann architectures, all configurations achieve approximately an 8× improvement in runtime.

**Off-chip energy:** Off-chip energy consumption exhibits a trend closely aligned with overall runtime. Although ReRAM exhibits approximately 0.8% higher energy consumption in the *IdlCAM* configuration, its superior FP ratio reduces the total number of memory interactions, thereby mitigating overall energy usage. In *EccCAM*, the FP ratios of STT-MRAM and ReRAM become comparable; consequently, STT-MRAM shows roughly

**Table 6.4.:** Simulation setup parameters/configurations.

| **Circuit-Level Parameters** | | | |
|---|---|---|---|
| STT-MRAM-Parameters | | ReRAM-Parameters | |
| Oxide Layer Thickness | 0.85 nm | Radius of Filament | 45 nm |
| Free Layer Thickness | 0.70 nm | Length of the Disc Region | 0.6 nm |
| MTJ Surface (circle) | r=40 nm | Initial Oxygen Vacancies Concentration in the Disc $[\times 10^{23}/m^3]$ | LRS: 3<br>HRS: 0.009 |
| $TMR_0$ (TMR at 0 V) | ~200% | | |
| Resistance-Area Product (RA) | 10 Ω µm$^2$ | Oxygen Vacancies Concentration in the Disc $[\times 10^{23}/m^3]$ | Max: 20<br>Min: 0.008 |
| LRS ($3\sigma$) | 3.1% | LRS ($3\sigma$) | 5.9% |
| HRS ($3\sigma$) | 8% | HRS ($3\sigma$) | 18% |
| 22 nm FD-SOI library (GlobalFoundries) | | | |

| **Extracted Circuit-Level Parameters** | | | | |
|---|---|---|---|---|
| | STT-MRAM-$\mu$ | STT-MRAM-$\sigma$ | ReRAM-$\mu$ | ReRAM-$\sigma$ |
| HRS | 23 kΩ | 1 kΩ | 250 kΩ | 45 kΩ |
| LRS | 12 kΩ | 200 Ω | 9.8 kΩ | 600 Ω |

| **Updated System-Level NVMain Configurations Source/Reference** | |
|---|---|
| STT-MRAM | ReRAM |
| DDR4-based 1 Gbit [223] | SPI-based 8 Mbit [253]<br>(extrapolated, validated against academic sources) |

| **System-Level gem5 Config file** | |
|---|---|
| CPU type | Out-of-Order |
| Compiler & Optimization Flag | g++ -O3 |
| ISA / CPU Clock Frequency | x86–64 / 2 GHz |
| # of Load/Store/Issue/ROB Entries | 32 / 32 / 64 / 128 |
| # of Physical Int/FP/Vec Registers | 128 / 128 / 128 |
| L1 Inst/Data Cache Size | 32 / 64 KiB |
| L1 Tag/Data/Response Latency | 2/2/2 cycles |
| L2 Cache Size | 2 MiB |
| L2 Tag/Data/Response Latency | 20/20/20 cycles |

| **Applications & Datasets** | |
|---|---|
| DB (Database Search) | Steam dataset [249] (small), myKet dataset [250] (large) |
| ISH (Image Similarity Hashing) | CIFAR-10 [251] |
| DNA (DNA Sequence Search) | Human genome data [252] |

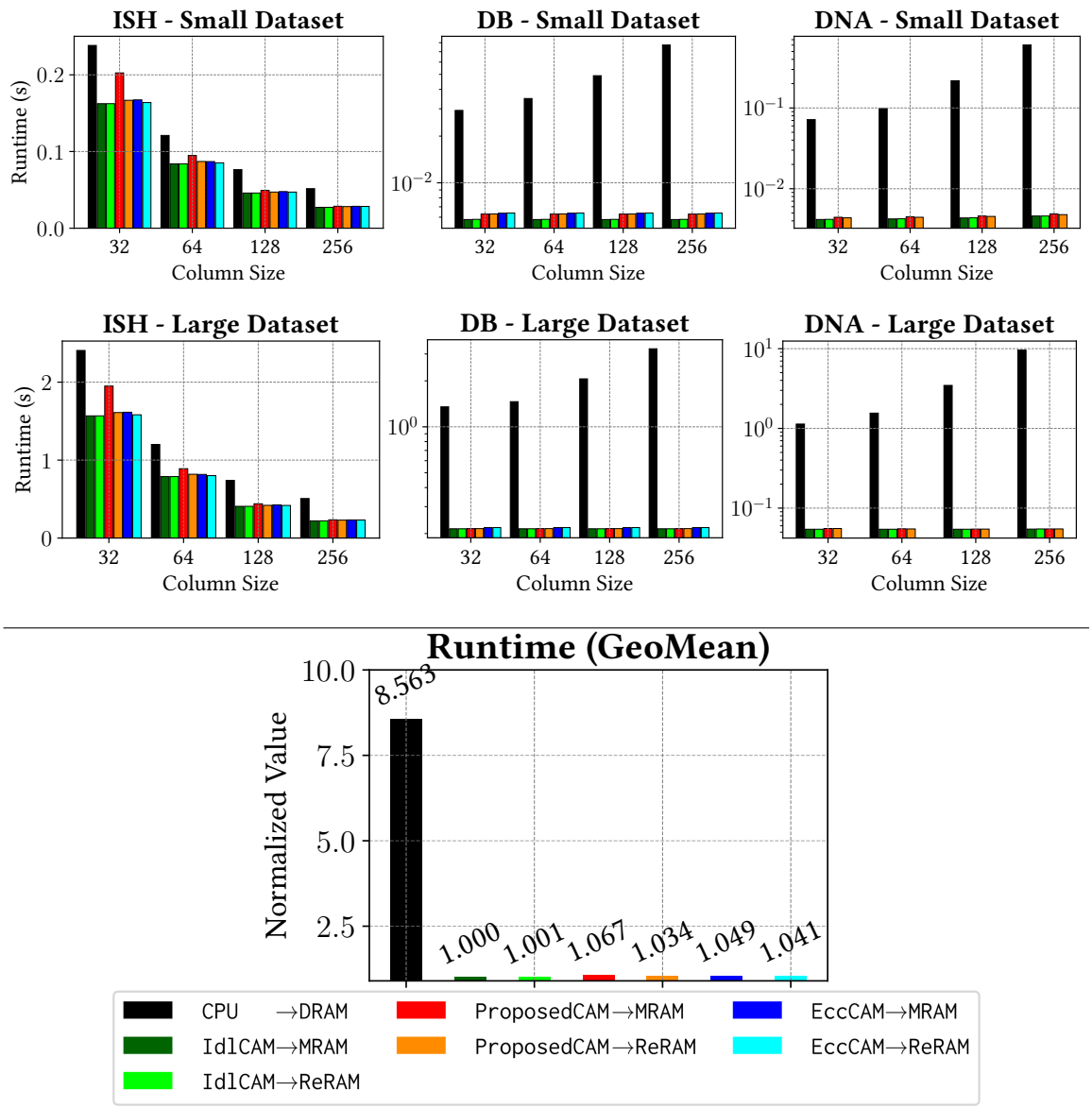



**Figure 6.7.:** Runtime across simulation configurations.

0.3% lower energy consumption, underscoring the impact of ECC decoding overhead in the ReRAM-based design. Overall, compared with conventional von Neumann architectures, all configurations achieve approximately a 2× improvement in off-chip energy efficiency.

**FP ratio:** FP ratio is primarily determined by dataset characteristics. Overall, STT-MRAM exhibits a highest FP ratio (~7×), consistent with its lower HRS-to-LRS ratio compared to ReRAM. However, results shows that ECC can drastically improve this to only 3%. Nevertheless, the system-level performance difference between STT-MRAM and ReRAM remains small, since the FP-to-dataset size ratio is minimal.

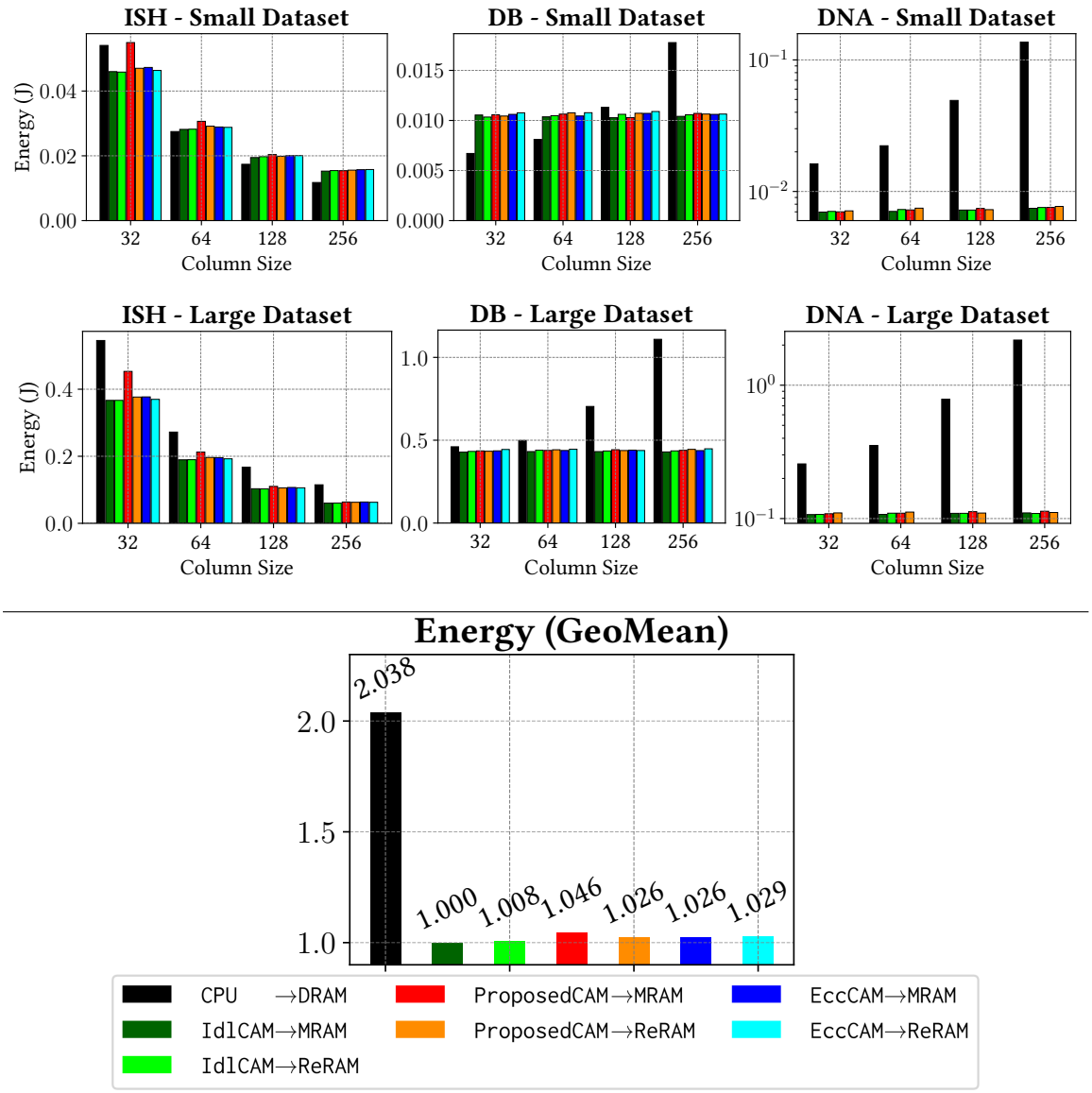



**Figure 6.8.:** Off-chip energy consumption across simulation configurations.

### 6.5.1. Approximate CAM Scenarios

To further highlight the flexibility of our framework for algorithm–technology co-optimization and approximate CAM scenarios, we implemented a k-Nearest Neighbors (k-NN) classification algorithm. To align with NVM-CAM-compatible operations, we used integer-encoded values in place of floating-point representations, enabling similarity evaluation via Hamming distance rather than cosine or Euclidean metrics. This design choice avoids architectural modifications or additional hardware support, demonstrating how approximate applications can natively exploit the proposed Framework.

For this experiment, we generated synthetic datasets tailored to evaluate k-NN under varying similarity thresholds. Figure 6.10 presents the runtime and energy consumption

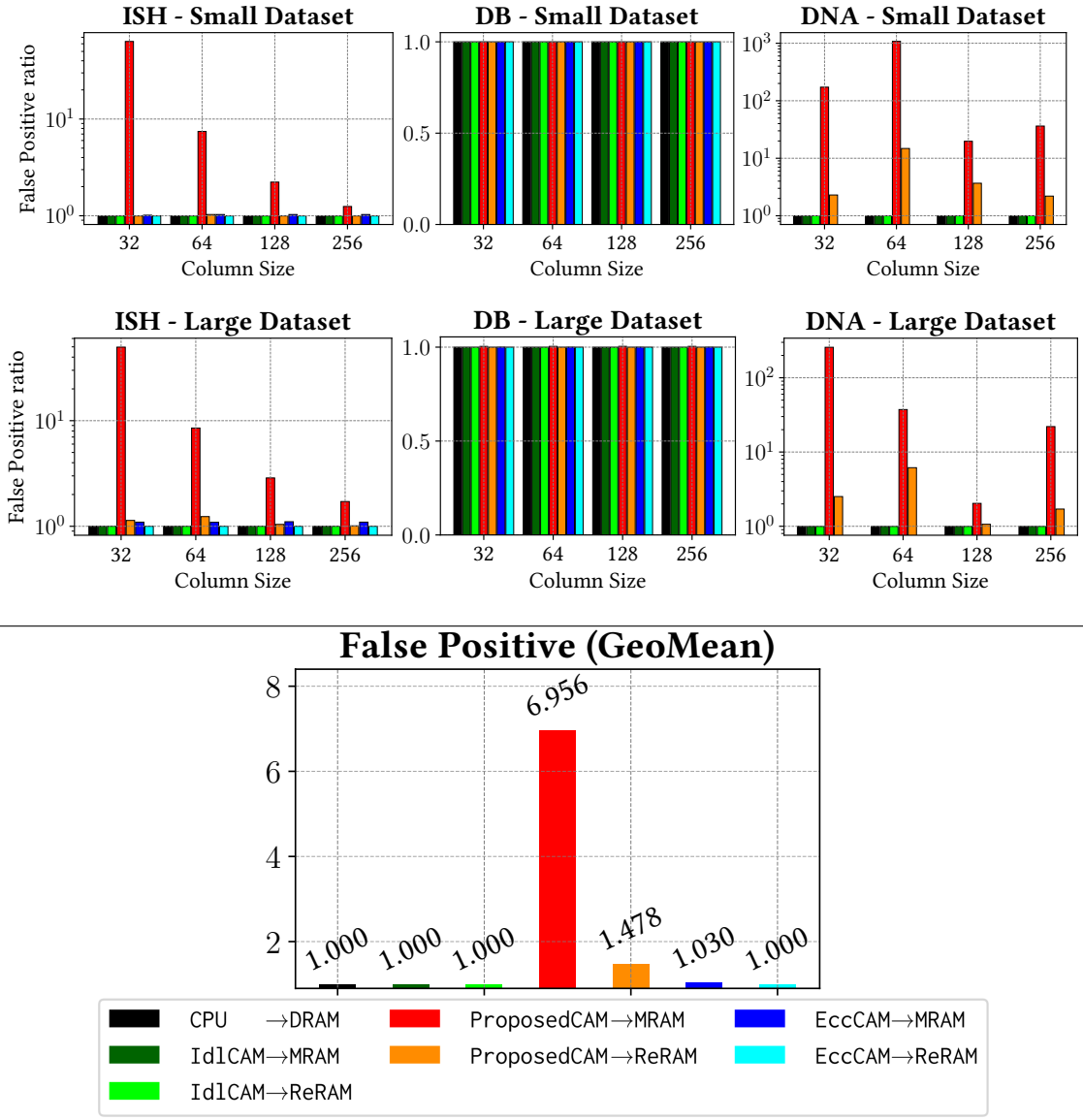



**Figure 6.9.:** False Positive Ratio (FP ratio) across simulation configurations.

for different configurations. Instead of reporting RF, we evaluate classification accuracy in Figure 6.11, since k-NN relies on majority voting rather than match validation. Here, in addition to the baseline $k$ values listed in Table 6.3, we incrementally increase $k$ by $\Delta k$ for each technology, effectively lowering $V_{\text{ref}}$ to explore broader similarity neighborhoods. The results suggest that STT-MRAM-based CAMs may be better suited for approximate workloads. Due to their lower HRS-to-LRS ratio and higher intrinsic variation, STT-MRAM devices introduce a broader range of match behavior, which can be beneficial for classification tasks that tolerate inexact matches.

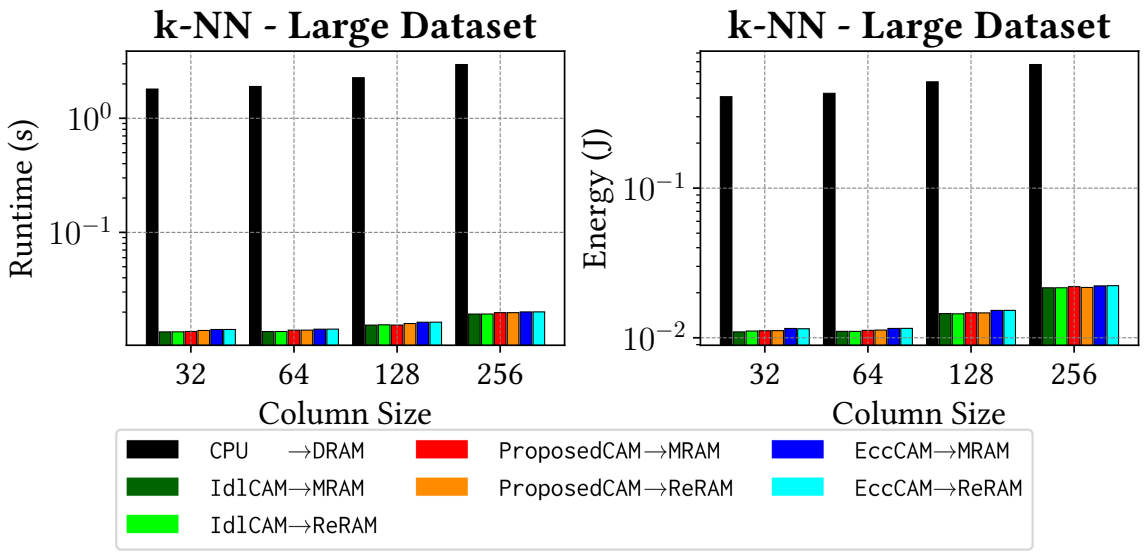



**Figure 6.10.:** Runtime and off-chip energy consumption for the k-NN classification task.

## 6.6. Conclusion

In this chapter, we introduced a cross-layer error-mitigation methodology for in-memory NVM-CAMs and validated it using an open-source simulation platform with detailed device-, circuit-, and architecture-level modeling. We showed that large-scale NVM-CAMs can serve as effective filtering accelerators for data-intensive, search-heavy applications, providing significant performance and off-chip energy-efficiency gains over conventional systems while guaranteeing zero SeER at the application level. Our evaluation further revealed that system-level behavior is largely insensitive to the specific NVM technology once appropriate algorithm–technology co-optimization is applied, because higher layers absorb many device-level differences. Realistic workloads confirm both the reliability and the practicality of the proposed error-handling strategy.

The flexibility of our platform enables systematic algorithm–technology co-optimization and naturally extends to hybrid memory hierarchies and near-memory compute units, broadening the applicability of reliable NVM-based search.

Together with the cross-layer techniques developed for NVM-CiM in the previous chapters, these results complete the core technical contributions of this dissertation. The concluding chapter reflects on these contributions, summarizes the overarching insights for reliable memory-centric computing, and outlines promising directions for future work.

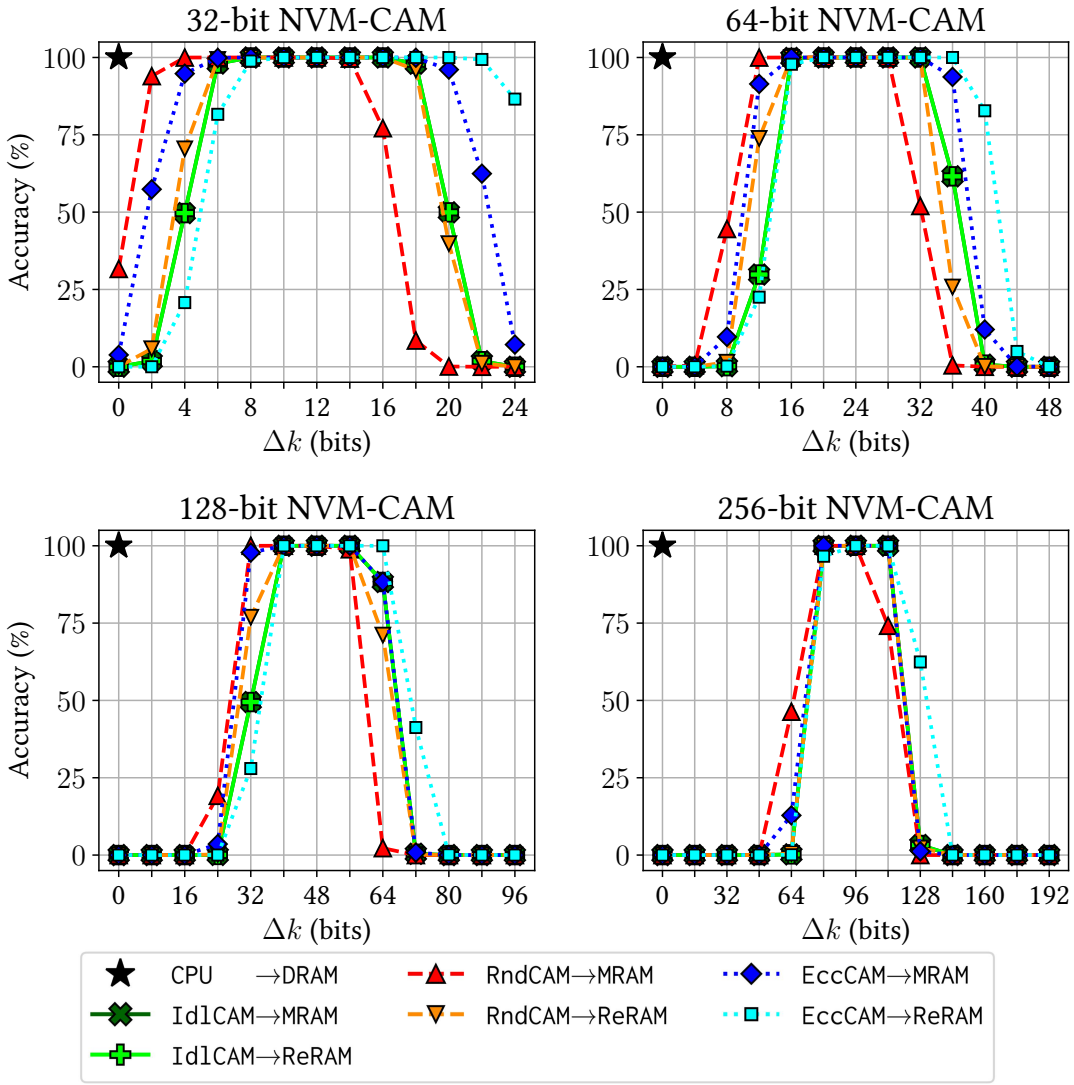



**Figure 6.11.:** Classification accuracy for the k-NN as $k$ is incrementally increased by $\Delta k$, effectively lowering $V_{\text{ref}}$.

# 7. Conclusion

This dissertation studied how to design, analyze, and improve the reliability of NVM-based in-memory computing substrates so that they provide performance and energy benefits with reliable behavior under realistic system constraints. The common theme is a cross-layer view: starting from device and circuit models, we propagate their impact through the microarchitecture and to the application level, and use this information to design error-mitigation mechanisms.

The work focused on two classes of memory-centric accelerators: ❶ NVM-CiM architectures, which embed Boolean or arithmetic operators directly into the memory arrays, and ❷ NVM-CAM structures, which perform associative search and similarity computation inside memory. Across these substrates, the dissertation pursued three main goals: ❶ build full-system modeling and fault-injection infrastructure; ❷ quantify the reliability, performance, and energy trade-offs of different technologies and architectural flavors under realistic workloads; and ❸ develop cross-layer techniques that compensate for device and circuit non-idealities while preserving the advantages of memory-centric computing.

In the following, we first summarize the contributions of each technical chapter, then discuss the cross-layer insights that follow from this work, and finally outline limitations and directions for future research.

## 7.1. Summary of Contributions

The dissertation is organized around four core technical chapters that build on the background in Chapter 2 and together address modeling, assessment, and cross-layer hardening for NVM-based in-memory computing.

### Full-System Framework for NVM-CiM Architectures (Chapter 3)

Chapter 3 introduced *Sisyphus*, a full-system simulation and fault-injection framework for NVM-CiM architectures. Sisyphus extends gem5 with configurable CiM modules based on STT-MRAM, ReRAM, and PCM, integrates detailed technology and circuit models via NVSim and McPAT, and provides a comprehensive API that allows real applications to invoke CiM operations. It further implements microarchitectural SFI for both the host CPU and the CiM units, enabling joint assessment of performance, energy, and reliability.

Using Sisyphus, we showed that CiM-enabled systems can achieve more than 7× latency improvement and around 19% power savings compared to conventional CPU-only systems for representative workloads. At the same time, we observed that these gains interact with reliability: different NVM technologies show different vulnerability profiles, and the resulting ERC trade-offs depend strongly on the workload. This chapter established the need for cross-layer models that explicitly incorporate reliability when evaluating memory-centric architectures.

## Cross-Layer Hardening of NVM-CiM (Chapter 4)

Building on the Sisyphus infrastructure, Chapter 4 addressed the reliability limitations of scouting-logic-based NVM-CiM operations. We showed that realistic process variations lead to circuit-level decision failure rates between $10^{-7}$ and $10^{-3}$, which is insufficient for large-scale deployment of NVM-CiM accelerators.

To overcome this bottleneck, we proposed a cross-layer software–hardware co-design based on *double-reference sensing* and selective recomputation. At the circuit level, each BL is sensed against two intentionally offset references, which partition the resistance space into error-free and suspicious regions. At the architectural level, suspicious results are flagged and propagated to the CPU; and at the application level, only the flagged subset is recomputed using conventional instructions.

Extensive full-system simulations with microarchitectural fault injection showed that this method reduces the application-level decision failure rate below $10^{-12}$, while preserving the performance and energy advantages of NVM-CiM. In particular, the proposed scheme delivers about 45% performance improvement and 30% energy savings over a CPU-only baseline, with roughly 1% verification overhead. This chapter shows that modest circuit changes, when combined with appropriate software support, can close the reliability gap of NVM-CiM without removing its benefits.

## Benchmarking NVM-CAM Similarity-Computing Flavors (Chapter 5)

While Chapter 3 and 4 concentrated on NVM-CiM, Chapter 5 turned to similarity computation using NVM-based CiM architectures that support either a scouting-based flavor or a dedicated CAM-based flavor. We developed an end-to-end technology-to-application analysis flow that combines SPICE-level characterization of NVM devices, circuit-level error modeling, architecture-level integration into gem5, and application-level error propagation via system-level Monte Carlo fault injection.

Using this framework, we benchmarked multiple CiM-friendly applications, including exact matching, DNA sequencing, hyperdimensional classification, image hashing, and Bloom filters, across three representative NVM technologies. The results showed that both similarity-computing flavors achieve latency improvements of 1.2×–3.4× and 2.2×–12×, and energy improvements up to 6.7×, relative to CPU-only execution. These gains are

largely technology-independent once full-system effects are taken into account. In contrast, reliability—captured via the AVF metric—is strongly technology- and flavor-dependent.

The chapter also showed that some applications, such as Bloom filters and image hashing, can benefit from the intrinsic non-idealities and randomness of NVM devices. In these cases, noise improves the entropy of hashing or index generation, which can lead to better false-positive characteristics compared to a noiseless CPU baseline. Overall, Chapter 5 quantified the reliability–performance–energy space of similarity-computing NVM-CiM flavors and identified application-dependent cases where non-ideal behavior is useful.

## Reliable NVM-CAM for Exact Matching (Chapter 6)

Finally, Chapter 6 addressed the challenge of providing *exact-match guarantees* in large-scale NVM-CAM arrays. Whereas Chapter 5 primarily targeted approximate similarity search, many data-intensive workloads (e.g., database lookups, key–value stores, and certain genomic queries) require strictly correct matching semantics.

The chapter proposed a cross-layer error-mitigation methodology that combines circuit-level tuning with algorithmic filtering. First, we statistically characterized the distribution of ML discharge behavior under process variation for STT-MRAM- and ReRAM-based 2T2R cells, and determined a technology- and configuration-dependent mismatch tolerance level $k$. Based on this analysis, we set a fixed, optimized reference voltage $V_{\text{ref}}$ that lies at the boundary between full matches and $k$-bit mismatches. This choice ensures that all true matches are always detected, at the cost of a controlled number of false positives.

Second, we modified the software search procedure so that the CPU verifies all candidates returned by the NVM-CAM. Because the number of candidates is typically much smaller than the dataset, this filtering step adds only modest overhead while guaranteeing zero SeER at the application level. Full-system simulations with realistic applications (e.g., database search, image similarity hashing, and DNA sequence search) showed that the proposed method incurs only about 6% runtime and 4% off-chip energy overhead compared to an ideal, error-free NVM-CAM, while still providing around 8× speedup and 2× off-chip energy savings over a conventional von Neumann baseline. The chapter also showed that, after appropriate co-optimization, system-level behavior becomes largely insensitive to the exact NVM technology, because higher layers absorb many device-level differences.

Together, these four technical chapters provide a consistent methodology: from full-system modeling and reliability assessment, through cross-layer protection of NVM-CiM, to benchmarking and error mitigation for NVM-CAM in both approximate and exact-match schemes.

## 7.2. Cross-Layer Insights for Reliable Memory-Centric Computing

Beyond the chapter-specific results, the dissertation gives several general observations about designing reliable memory-centric architectures.

**End-to-end modeling is necessary.** Device-level or array-level metrics alone do not predict system-level behavior well. For example, ReRAM appears more robust than STT-MRAM at the device level, yet the effective reliability and performance in a full system depend on how technology characteristics interact with cache hierarchies, data movement patterns, and application-level masking. Likewise, some technologies with less favorable raw parameters can be competitive when paired with appropriate architectural and algorithmic support. Cross-layer models are therefore important when comparing technologies or choosing design points.

**Reliability must be a design objective.** In all evaluated scenarios, designs that maximize performance or minimize energy without considering reliability are not suitable for large-scale deployment. The value of CiM and CAM-style accelerators lies not only in their raw speedups but also in their ability to deliver those speedups within acceptable failure rates and without prohibitive protection overheads. Explicitly quantifying the energy–reliability–cost trade-offs, as done via the ERC and AVF analyses, helps to identify useful combinations of technology, architecture, and workload.

**Cross-layer co-design enables practical error mitigation.** The proposed schemes in Chapter 4 and Chapter 6 follow a common pattern: use relatively simple circuit or architectural mechanisms to expose potentially unreliable events, and offload the rare, critical corrections to more flexible software layers. This division of responsibilities keeps hardware overheads low while exploiting the programmability and context knowledge of software. By controlling when and how software intervenes, it is possible to reach very low or even zero error rates at minimum cost.

**Non-idealities are not always harmful.** While much of this dissertation focuses on mitigating errors, Chapter 5 shows that, for some approximate applications, device non-idealities can be beneficial. In image hashing and Bloom-filter-like scenarios, NVM variability introduces additional randomness that improves hash quality or mitigates unintended results. This suggests that, for some workloads, error-mitigation mechanisms and hardware parameters should not only minimize errors, but also shape non-ideal behaviors toward application-level goals.

## 7.3. Future Research Directions

The methodology and insights developed in this dissertation lead to several directions for future work.

**Extending modeling frameworks to richer memory hierarchies.** One natural extension is to adapt Sisyphus and the NVM-CAM platform to heterogeneous memory systems that combine DRAM, SRAM, multiple NVM tiers, and emerging interfaces such as 3D-stacked memories and CXL-attached devices. Such systems introduce new data-movement patterns and reliability modes that require corresponding cross-layer models and protection strategies.

**Generalizing cross-layer reliability to other accelerators.** The co-design patterns used here are not specific to NVM-based memories. Analog and mixed-signal accelerators for machine learning, in-sensor processing, or near-storage computation face similar challenges. Adapting the modeling and error-mitigation concepts, including selective recomputation, candidate verification, and reliability-aware scheduling, to these domains could broaden the impact of this work.

**Adaptive and workload-aware error management.** Both the double-reference sensing scheme and the tuned-$V_{\text{ref}}$ NVM-CAM design use static design-time parameters. An extension is to make these parameters adaptive, for example by adjusting reference levels or verification policies based on observed workload characteristics, environmental conditions, or aging indicators. Such adaptive schemes could further reduce overheads while maintaining target reliability.

**Software and programming-model support.** Fully realizing the potential of reliable memory-centric computing will also require better integration with software. This includes compiler and runtime support for automatically offloading suitable kernels to CiM or CAM substrates, annotating reliability requirements, and coordinating error-mitigation policies across libraries and OS.

# Closing Remarks

In summary, this dissertation has shown that NVM-based in-memory computing can provide substantial performance and energy benefits while meeting strict reliability targets, if it is designed and evaluated with a cross-layer perspective. Accurate end-to-end modeling, combined with targeted error-mitigation mechanisms, allows device- and circuit-level non-idealities to be treated as a design factor rather than a fundamental limitation. The

techniques and insights developed here contribute to making memory-centric computing a practical and reliable option in future systems.

# Part III.

# Appendix

# List of Acronyms

| | |
|---|---|
| **ACE** | Architecturally Correct Execution |
| **ADC** | Analog-to-Digital Converter |
| **ALU** | Arithmetic Logic Unit |
| **AM** | Associative Memory |
| **API** | Application Programming Interface |
| **AVF** | Architectural Vulnerability Factor |
| **BER** | Bit Error Rate |
| **BL** | Bit Line |
| **CAM** | Content Addressable Memory |
| **CDF** | Cumulative Distribution Function |
| **CiM** | Compute-in-Memory |
| **CMOS** | Complementary Metal-Oxide-Semiconductor |
| **CPU** | Central Processing Unit |
| **DDR4** | double data rate fourth-generation |
| **DMA** | Direct Memory Access |
| **DNA** | Deoxyribonucleic Acid |
| **DRAM** | Dynamic Random-Access Memory |
| **ECC** | Error-Correcting Code (ECC) |
| **ED2P** | Energy-Delay Squared Product |
| **EDP** | Energy-Delay Product |
| **ERC** | Efficiency-Resilience Coefficient |
| **FIT** | Failures in Time |
| **FPE** | Failure per Execution |
| **FPGA** | Field-Programmable Gate Array |
| **HBM** | High-Bandwidth Memory |
| **HDC** | HyperDimensional Computing |
| **HMC** | Hybrid Memory Cube |
| **HRS** | high-resistive state |
| **IID** | Independent and Identically Distributed |
| **IP** | Intellectual Property |
| **IQ** | Instruction Queue |
| **IR** | Intermediate Representation |
| **ISA** | Instruction Set Architecture |
| **k-NN** | k-Nearest Neighbors |
| **LRDIMM** | Load-Reduced Dual Inline Memory Module |
| **LRS** | low-resistive state |

| | |
|---|---|
| **MAC** | Multiply–Accumulate |
| **ML** | Match Line |
| **MTJ** | Magnetic Tunnel Junction |
| **MVM** | Matrix–Vector Multiplication |
| **NMC** | Near-Memory Computing |
| **NMOS** | N-type Metal-Oxide-Semiconductor |
| **NN** | Neural Network |
| **NVM** | Non-Volatile Memory |
| **OS** | Operating System |
| **PCM** | Phase Change Memory |
| **PDF** | Probability Density Function |
| **PDK** | Process Design Kit |
| **PDN** | Power Distribution Network |
| **PiM** | Processing-in-Memory |
| **PMA** | Perpendicular Magnetic Anisotropy |
| **ReRAM** | Redox-based RAM |
| **ROB** | Reorder Buffer |
| **RTL** | Register-Transfer Level |
| **SA** | Sense Amplifier |
| **SBU** | Single-Bit-Upset |
| **SDC** | Silent Data Corruption |
| **SECDED** | Single Error Correction, Double Error Detection |
| **SeER** | Search Error Rate |
| **SFI** | Statistical Fault Injection |
| **SIMD** | Single Instruction Multiple Data |
| **SL** | Source Line |
| **SoC** | System-on-Chip |
| **SRAM** | Static Random-Access Memory |
| **STT-MRAM** | Spin-Transfer Torque Magnetic RAM |
| **TLB** | Translation-Lookaside Buffer |
| **WL** | Word Line |

# List of Figures

# List of Tables

# List of Algorithms

# Listings

# Bibliography

*The titles of most entries are hyperlinks resolving the DOIs or pointing to other online sources for the entries.*


[1] A. Nezhadi, O. Chatzopoulos, M. Mayahinia, G. Papadimitriou, M. Tahoori, and D. Gizopoulos. "Sisyphus: Cross-Layer Efficiency Across NVM Technologies in Compute-in-Memory Architectures". In: *2025 IEEE International Test Conference (ITC)*. 2025, pp. 213–222.

[2] A. Nezhadi, O. Chatzopoulos, D. Gizopoulos, and M. Tahoori. "Trust, but Verify: Reliable Compute-in-Memory via Double-Reference Sensing and Selective Recompute". In: *Accepted at IOLTS 2026 Conference*. 2026.

[3] A. Nezhadi, M. Mayahinia, and M. Tahoori. "Cross-Layer Reliability Evaluation of In-Memory Similarity Computation". In: *2024 IEEE International Test Conference (ITC)*. 2024, pp. 81–85.

[4] A. Nezhadi, S. B. Mamaghani, and M. Tahoori. "Algorithm–Technology Co-Optimization for Reliable NVM-CAM Systems". In: *2026 IEEE 44th VLSI Test Symposium (VTS)*. 2026, pp. 1–7.

[5] S. M. Siddaramu, A. Nezhadi, M. Mayahinia, S. Ghasemi, and M. B. Tahoori. "Hardware and Software Co-Design for Optimized Decoding Schemes and Application Mapping in NVM Compute-in-Memory Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3744–3755.

[6] S. M. Siddaramu, A. Nezhadi, M. Mayahinia, S. Ozev, and M. B. Tahoori. "Temporal Reference Scouting Logic for PVT Reliable Logic Computation-in-Memory". In: *2026 IEEE 44th VLSI Test Symposium (VTS)*. 2026, pp. 1–7.

[7] H. Farzaneh, J. P. De Lima, A. Nezhadi Khelejani, A. A. Khan, M. Mayahinia, M. Tahoori, and J. Castrillon. "SHERLOCK: Scheduling Efficient and Reliable Bulk Bitwise Operations in NVMs". In: *Proceedings of the 61st ACM/IEEE Design Automation Conference*. DAC '24. Association for Computing Machinery, 2024.

[8] Y.-C. Chen, T. Seidl, N. Hölscher, C. Hakert, M. D. Truong, J.-J. Chen, J. P. C. de Lima, A. A. Khan, J. Castrillon, A. Nezhadi, L. Siddhu, H. Nassar, M. Mayahinia, M. B. Tahoori, J. Henkel, N. Wilbert, S. Wildermann, and J. Teich. *Modeling and Simulating Emerging Memory Technologies: A Tutorial*. 2025. arXiv: 2502.10167.

[9] J. von Neumann. "First draft of a report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75.

[10] J. L. Hennessy and D. A. Patterson. "A New Golden Age for Computer Architecture". In: *Commun. ACM* 62.2 (2019), pp. 48–60.

[11] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie. "Quantifying the energy cost of data movement in scientific applications". In: *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE. 2013, pp. 56–65.

[12] D. Pandiyan and C.-J. Wu. "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms". In: *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2014, pp. 171–180.

[13] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, et al. "Google workloads for consumer devices: Mitigating data movement bottlenecks". In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 2018, pp. 316–331.

[14] W. Dally. "Challenges for future computing systems". In: *HiPEAC keynote* (2015), pp. 1–66.

[15] K. K. Chang. "Understanding and improving the latency of DRAM-based memory systems". PhD thesis. Carnegie Mellon University, 2017.

[16] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. "Disaggregated memory for expansion and sharing in blade servers". In: *ACM SIGARCH computer architecture news* 37.3 (2009), pp. 267–278.

[17] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms". In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), pp. 60–71.

[18] Nuvation Engineering. *HMC 30G VSR Specification*. Nuvation Engineering. Accessed: 2024-10-17. 2024.

[19] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim. "Hbm (high bandwidth memory) dram technology and architecture". In: *2017 IEEE International Memory Workshop (IMW)*. IEEE. 2017, pp. 1–4.

[20] G. H. Loh. "3D-stacked memory architectures for multi-core processors". In: *ACM SIGARCH computer architecture news* 36.3 (2008), pp. 453–464.

[21] C. Weis, M. Jung, and N. Wehn. "3D stacked DRAM memories". In: *Handbook of 3D Integration: Design, Test, and Thermal Management* 4 (2019), pp. 149–186.

[22] T. Zhang, K. Wang, Y. Feng, X. Song, L. Duan, Y. Xie, X. Cheng, and Y.-L. Lin. "A customized design of DRAM controller for on-chip 3D DRAM stacking". In: *IEEE Custom Integrated Circuits Conference 2010*. IEEE. 2010, pp. 1–4.

[23] C. Weis, N. Wehn, L. Igor, and L. Benini. "Design space exploration for 3D-stacked DRAMs". In: *2011 Design, Automation & Test in Europe*. IEEE. 2011, pp. 1–6.

[24] Y.-D. Chih, Y.-C. Shih, C.-F. Lee, Y.-A. Chang, P.-H. Lee, H.-J. Lin, Y.-L. Chen, C.-P. Lo, M.-C. Shih, K.-H. Shen, H. Chuang, and T.-Y. J. Chang. "13.3 A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150°C and High Immunity to Magnetic Field Interference". In: *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2020, pp. 222–224.

[25] K. Lee, D. S. Kim, J. H. Bak, S. P. Ko, W. C. Lim, H. C. Shin, J. H. Lee, J. H. Park, J. H. Jeong, J. M. Lee, T. Kai, H. Sato, J. W. Lee, K. H. Ryu, Y. J. Kim, S. H. Han, B. Y. Seo, K. S. Suh, H. H. Kim, H. T. Jung, D. H. Jang, N. Y. Ji, M. J. Eom, I. H. Kim, K. Lee, K. H. Hwang, Y. J. Song, and H. S. Kim. "28nm CIS-Compatible Embedded STT-MRAM for Frame Buffer Memory". In: *2021 IEEE International Electron Devices Meeting (IEDM)*. 2021, pp. 2.1.1–2.1.4.

[26] M.-F. Chang, J.-J. Wu, T.-F. Chien, Y.-C. Liu, T.-C. Yang, W.-C. Shen, Y.-C. King, C.-J. Lin, K.-F. Lin, Y.-D. Chih, S. Natarajan, and J. Chang. "19.4 embedded 1Mb ReRAM in 28nm CMOS with 0.27-to-1V read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014, pp. 332–333.

[27] W. C. Chien, L. M. Gignac, Y. C. Chou, C. H. Yang, N. Gong, H. Y. Ho, C. W. Yeh, H. Y. Cheng, W. Kim, I. T. Kuo, E. K. Lai, C. W. Cheng, L. Buzi, A. Ray, C. S. Hsu, D. Daudelin, R. L. Bruce, M. BrightSky, and H. L. Lung. "Device Study on OTS-PCM for Persistent Memory Application : IBM/Macronix Phase Change Memory Joint Project". In: *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. 2022, pp. 327–329.

[28] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. "Processing data where it makes sense: Enabling in-memory computation". In: *Microprocessors and Microsystems* 67 (2019), pp. 28–41.

[29] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. "A Modern Primer on Processing in Memory". In: *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer Nature Singapore, 2023, pp. 171–243.

[30] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu. "Processing-in-memory: A workload-driven perspective". In: *IBM Journal of Research and Development* 63.6 (2019), pp. 3–1.

[31] A. Gebregiorgis, H. A. Du Nguyen, J. Yu, R. Bishnoi, M. Taouil, F. Catthoor, and S. Hamdioui. "A survey on memory-centric computer architectures". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 18.4 (2022), pp. 1–50.

[32] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, and J. S. Vetter. "A survey on processing-in-memory techniques: Advances and challenges". In: *Memories-Materials, Devices, Circuits and Systems* 4 (2023), p. 100022.

[33] W. Haensch, A. Raghunathan, K. Roy, B. Chakrabarti, C. M. Phatak, C. Wang, and S. Guha. "Compute in-memory with non-volatile elements for neural networks: A review from a co-design perspective". In: *Advanced Materials* 35.37 (2023), p. 2204944.

[34] S. Mittal. "A survey of ReRAM-based architectures for processing-in-memory and neural networks". In: *Machine learning and knowledge extraction* 1.1 (2018), pp. 75–114.

[35] D. Milojicic, K. Bresniker, G. Campbell, P. Faraboschi, J. P. Strachan, and S. Williams. "Computing in-memory, revisited". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1300–1309.

[36] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, et al. "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization". In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. 2013, pp. 185–197.

[37] K. Chmielewski, J. Ławnicki, U. Lukyanau, T. Kobus, and M. Maciejewski. *UPMEM Unleashed: Software Secrets for Speed*. 2025. arXiv: 2510.15927 [cs.AR].

[38] B. Friesel, M. Lütke Dreimann, and O. Spinczyk. "A Full-System Perspective on UPMEM Performance". In: *Proceedings of the 1st Workshop on Disruptive Memory Systems*. DIMES '23. Association for Computing Machinery, 2023, pp. 1–7.

[39] K. Stockinger, J. Cieslewicz, K. Wu, D. Rotem, and A. Shoshani. "Using bitmap index for joint queries on structured and text data". In: *New Trends in Data Warehousing and Data Analysis*. Springer, 2008, pp. 1–23.

[40] Y. Li and J. M. Patel. "Bitweaving: Fast scans for main memory data processing". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, pp. 289–300.

[41] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. "Basic local alignment search tool". In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[42] H. Xin, J. Greth, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu. "Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping". In: *Bioinformatics* 31.10 (2015), pp. 1553–1560.

[43] F. Y. Shih. *Image processing and mathematical morphology: fundamentals and applications*. CRC press, 2017.

[44] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *Seminal graphics: pioneering efforts that shaped the field*. 1998, pp. 347–353.

[45] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das. "Cache automaton". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-50 '17. Association for Computing Machinery, 2017, pp. 259–272.

[46] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 2017, pp. 273–287.

[47] S. Li, D. Niu, B. J. Lee, C. R. Wu, and Y. Chen. "DRISA: A DRAM-based reconfigurable in-situ accelerator". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 288–301.

[48] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan. "Computing in memory with spin-transfer torque magnetic RAM". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.3 (2017), pp. 470–483.

[49] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman. "Ac-dimm: associative computing with stt-mram". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 2013, pp. 189–200.

[50] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 14–26.

[51] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. "MAGIC–memristor-aided logic". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 61.11 (2014), pp. 895–899.

[52] S. Li, D. Niu, K.-T. Ho, K. Kim, R. Karri, M. P. Frank, and Y. Chen. "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories". In: *Proceedings of the 53rd Annual Design Automation Conference (DAC)*. ACM, 2016, pp. 1–6.

[53] M. L. Gallo, A. Sebastian, G. Cherubini, H. Giefers, and E. Eleftheriou. "Mixed-precision in-memory computing". In: *Nature Electronics* 1.4 (2018), pp. 246–253.

[54] S. R. S. Raman. "A Review on Non-Volatile and Volatile Emerging Memory Technologies". In: *Computer Memory and Data Storage*. IntechOpen, 2024. Chap. 3.

[55] D. Ielmini and H.-S. P. Wong. "In-memory computing with resistive switching devices". In: *Nature Electronics* 1.6 (2018), pp. 333–343.

[56] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. B. Tahoori. "Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.3 (2015), pp. 367–380.

[57] B. Li, B. Yan, and H. Li. "An overview of in-memory processing with emerging non-volatile memory for data-intensive applications". In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. 2019, pp. 381–386.

[58] M. Giordano, K. Prabhu, K. Koul, R. M. Radway, A. Gural, R. Doshi, Z. F. Khan, J. W. Kustin, T. Liu, G. B. Lopes, V. Turbiner, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, G. Lallement, B. Murmann, S. Mitra, and P. Raina. "CHIMERA: A 0.92 TOPS, 2.2 TOPS/W Edge AI Accelerator with 2 MByte On-Chip Foundry Resistive RAM for Efficient Training and Inference". In: *2021 Symposium on VLSI Circuits*. 2021, pp. 1–2.

[59] W.-H. Huang, T.-H. Wen, J.-M. Hung, W.-S. Khwa, Y.-C. Lo, C.-J. Jhang, H.-H. Hsu, Y.-H. Chin, Y.-C. Chen, C.-C. Lo, R.-S. Liu, K.-T. Tang, C.-C. Hsieh, Y.-D. Chih, T.-Y. Chang, and M.-F. Chang. "A Nonvolatile Al-Edge Processor with 4MB SLC-MLC Hybrid-Mode ReRAM Compute-in-Memory Macro and 51.4-251TOPS/W". In: *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. 2023, pp. 15–17.

[60] S. D. Spetalnick, M. Chang, S. Konno, B. Crafton, A. S. Lele, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury. "A 40-nm Compute-in-Memory Macro With RRAM Addressing IR Drop and Off-State Current". In: *IEEE Solid-State Circuits Letters* 7 (2024), pp. 10–13.

[61] Y.-C. Chiu, W.-S. Khwa, C.-Y. Li, F.-L. Hsieh, Y.-A. Chien, G.-Y. Lin, P.-J. Chen, T.-H. Pan, D.-Q. You, F.-Y. Chen, A. Lee, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, Y.-D. Chih, T.-Y. Chang, and M.-F. Chang. "A 22nm 8Mb STT-MRAM Near-Memory-Computing Macro with 8b-Precision and 46.4-160.1TOPS/W for Edge-AI Devices". In: *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. 2023, pp. 496–498.

[62] W. S. Khwa, K. Akarvardar, Y. S. Chen, Y. C. Chiu, J. C. Liu, J. J. Wu, H. Y. Lee, S. M. Yu, C. H. Lee, T. C. Chen, Y. C. Lin, C. F. Hsu, T. Y. Lee, T. K. Ku, C. H. Kuo, J. Y. Wu, X. Y. Bao, C. S. Chang, Y. D. Chih, H.-S. P. Wong, and M. F. Chang. "MLC PCM Techniques to Improve Nerual Network Inference Retention Time by 105X and Reduce Accuracy Degradation by 10.8X". In: *2021 Symposium on VLSI Technology*. 2021, pp. 1–2.

[63] S. Jung, H. Lee, S. Myung, H. Kim, S. K. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, et al. "A crossbar array of magnetoresistive memory devices for in-memory computing". In: *Nature* 601.7892 (2022), pp. 211–216.

[64] G. S. Syed, K. Brew, A. Vasilopoulos, V. P. Jonnalagadda, B. Kersting, T. Philip, V. Bragaglia, S. Ambrogio, J. Büchel, J. Giannopoulos, M. L. Gallo, C.-W. Cheng, M. BrightSky, V. Narayanan, N. Saulnier, and A. Sebastian. "In-Memory Compute Chips with Carbon-based Projected Phase-Change Memory Devices". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, pp. 1–4.

[65] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy. "Challenges for high-density 16Gb ReRAM with 27nm technology". In: *2015 Symposium on VLSI Technology (VLSI Technology)*. 2015, T106–T107.

[66] I. Alam, S. Pal, and P. Gupta. "Compression with multi-ECC: enhanced error resiliency for magnetic memories". In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS '19. Association for Computing Machinery, 2019, pp. 85–100.

[67] N. H. Seong, S. Yeo, and H.-H. S. Lee. "Tri-level-cell phase change memory: toward an efficient and reliable memory system". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA '13. Association for Computing Machinery, 2013, pp. 440–451.

[68] Y. Emre, C. Yang, K. Sutaria, Y. Cao, and C. Chakrabarti. "Enhancing the Reliability of STT-RAM through Circuit and System Level Techniques". In: *2012 IEEE Workshop on Signal Processing Systems*. 2012, pp. 125–130.

[69] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi. "An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors". In: *IEEE Transactions on Parallel and Distributed Systems* 28.6 (2017), pp. 1564–1577.

[70] R. Karam, R. Puri, S. Ghosh, and S. Bhunia. "Emerging trends in design and applications of memory-based computing and content-addressable memories". In: *Proceedings of the IEEE* 103.8 (2015), pp. 1311–1330.

[71] T. Molom-Ochir, B. Taylor, H. Li, and Y. Chen. "Advancements in Content-Addressable Memory (CAM) Circuits: State-of-the-Art, Applications, and Future Directions in the AI Domain". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2025).

[72] K. Pagiamtzis and A. Sheikholeslami. "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey". In: *IEEE journal of solid-state circuits* 41.3 (2006), pp. 712–727.

[73] C. E. Graves, C. Li, G. Pedretti, and J. P. Strachan. "In-memory computing with non-volatile memristor cam circuits". In: *Memristor Computing Systems*. Springer, 2022, pp. 105–139.

[74] C. E. Graves, C. Li, X. Sheng, D. Miller, J. Ignowski, L. Kiyama, and J. P. Strachan. "In-memory computing with memristor content addressable memories for pattern matching". In: *Advanced Materials* 32.37 (2020), p. 2003437.

[75] C. Li, C. E. Graves, X. Sheng, D. Miller, M. Foltin, G. Pedretti, and J. P. Strachan. "Analog content-addressable memories with memristors". In: *Nature communications* 11.1 (2020), p. 1638.

[76] G. Pedretti, C. E. Graves, T. Van Vaerenbergh, S. Serebryakov, M. Foltin, X. Sheng, R. Mao, C. Li, and J. P. Strachan. "Differentiable content addressable memory with memristors". In: *Advanced electronic materials* 8.8 (2022), p. 2101198.

[77] H. Zhong, S. Cao, H. Yang, and X. Li. "Dynamic ternary content-addressable memory is indeed promising: Design and benchmarking using nanoelectromechanical relays". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1100–1103.

[78] M. Mayahinia, S. Thomann, P. Genssler, C. Münch, H. Amrouch, and M. Tahoori. "Algorithm to Technology Co-Optimization for CiM-based Hyperdimensional Computing". In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024.

[79] K.-J. Zhou, C. Mu, B. Wen, X.-M. Zhang, G.-J. Wu, C. Li, H. Jiang, X.-Y. Xue, S. Tang, C.-X. Chen, et al. "The trend of emerging non-volatile TCAM for parallel search and AI applications". In: *Chip* 1.2 (2022), p. 100012.

[80] M. Li, S. Liu, M. M. Sharifi, and X. S. Hu. "CAMASim: A Comprehensive Simulation Framework for Content-Addressable Memory based Accelerators". In: *arXiv* (2024).

[81] C. Li, F. Müller, T. Ali, R. Olivo, M. Imani, S. Deng, C. Zhuo, T. Kämpfe, X. Yin, and K. Ni. "A scalable design of multi-bit ferroelectric content addressable memory for data-centric computing". In: *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2020, pp. 29–3.

[82] H. Caminal, K. Yang, S. Srinivasa, A. K. Ramanathan, K. Al-Hawaj, T. Wu, V. Narayanan, C. Batten, and J. F. Martínez. "CAPE: A content-addressable processing engine". In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2021, pp. 557–569.

[83] M. E. Kanakis, R. Khalili, and L. Wang. "Machine learning for computer systems and networking: A survey". In: *ACM Computing Surveys* 55.4 (2022), pp. 1–36.

[84] T. Macedo and F. Oliveira. *Redis cookbook: Practical techniques for fast data manipulation*. " O'Reilly Media, Inc.", 2011.

[85] B. Fitzpatrick. "Distributed caching with memcached". In: *Linux journal* 2004.124 (2004), p. 5.

[86] M. Zheludkov, T. Isachenko, et al. *High Performance in-memory computing with Apache Ignite*. Lulu. com, 2017.

[87] Y. Wang, G. Zhong, L. Kun, L. Wang, H. Kai, F. Guo, C. Liu, and X. Dong. "The performance survey of in memory database". In: *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2015, pp. 815–820.

[88] R. Hanhan, E. Garzón, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits. "EDAM: Edit distance tolerant approximate matching content addressable memory". In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 2022, pp. 495–507.

[89] Z. Jahshan, I. Merlin, E. Garzón, and L. Yavits. "Dash-cam: Dynamic approximate search content addressable memory for genome classification". In: *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 2023, pp. 1453–1465.

[90] B. Crafton, S. D. Spetalnick, M. Chang, and A. Raychowdhury. "A 28nm approximate/binary 6T CAM for sequence alignment". In: *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE. 2024, pp. 1–2.

[91] E. Garzón, R. Golman, Z. Jahshan, R. Hanhan, N. Vinshtok-Melnik, M. Lanuzza, A. Teman, and L. Yavits. "Hamming distance tolerant content-addressable memory (HD-CAM) for DNA classification". In: *IEEE Access* 10 (2022), pp. 28080–28093.

[92] S. K. Khatamifard, Z. Chowdhury, N. Pande, M. Razaviyayn, C. Kim, and U. R. Karpuzcu. *Read Mapping Near Non-Volatile Memory*. 2020. arXiv: `1709.02381 [cs.DC]`.

[93] E. Garzón, L. Yavits, A. Teman, and M. Lanuzza. "Approximate content-addressable memories: a review". In: *Chips* 2.2 (2023), pp. 70–82.

[94] P. Indyk and R. Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*. 1998, pp. 604–613.

[95] H. Jégou, M. Douze, and C. Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128.

[96] B. H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[97] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. 2nd. Addison-Wesley, 1998.

[98] S. Narla, P. Kumar, M. Adnaan, and A. Naeemi. *Cross-layer Modeling and Design of Content Addressable Memories in Advanced Technology Nodes for Similarity Search*. 2024. arXiv: 2403.15328 [cs.ET].

[99] E. Garzón, L. Yavits, G. Finocchio, M. Carpentieri, A. Teman, and M. Lanuzza. "A low-energy DMTJ-based ternary content-addressable memory with reliable sub-nanosecond search operation". In: *IEEE Access* 11 (2023), pp. 16812–16819.

[100] B. Song, T. Na, J. P. Kim, S. H. Kang, and S.-O. Jung. "A 10T-4MTJ nonvolatile ternary CAM cell for reliable search operation and a compact area". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 64.6 (2016), pp. 700–704.

[101] A. P. Chavan et al. "Design of Magnetic Tunnel Junction based Low Power Non-Volatile Ternary Content Addressable Memory". In: *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*. IEEE. 2019, pp. 1007–1011.

[102] A. Shaban, S. Ahmad, N. Alam, and M. Hasan. "Compact and Reliable Low Power Non-Volatile TCAM Cell". In: *2018 8th International Symposium on Embedded Computing and System Design (ISED)*. IEEE. 2018, pp. 100–104.

[103] D. Cho, K. Kim, and C. Yoo. "A non-volatile ternary content-addressable memory cell for low-power and variation-toleration operation". In: *IEEE transactions on magnetics* 54.2 (2017), pp. 1–3.

[104] C. Wang, D. Zhang, L. Zeng, E. Deng, J. Chen, and W. Zhao. "A novel MTJ-based non-volatile ternary content-addressable memory for high-speed, low-power, and high-reliable search operation". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.4 (2018), pp. 1454–1464.

[105] X. Wang, D. Zhang, K. Zhang, E. Deng, Y. Wang, and W. Zhao. "A novel multi-context non-volatile content-addressable memory cell and multi-level architecture for high reliability and density". In: *2021 IEEE 10th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE. 2021, pp. 1–6.

[106] C. Wang, D. Zhang, L. Zeng, and W. Zhao. "Design of magnetic non-volatile TCAM with priority-decision in memory technology for high speed, low power, and high reliability". In: *IEEE transactions on circuits and systems I: regular papers* 67.2 (2019), pp. 464–474.

[107] S. Narla, P. Kumar, A. F. Laguna, D. Reis, X. S. Hu, M. Niemier, and A. Naeemi. "Design of a Compact Spin-Orbit-Torque-Based Ternary Content Addressable Memory". In: *IEEE Transactions on Electron Devices* 70.2 (2023), pp. 506–513.

[108] D. Cho, K. Kim, and C. Yoo. "Variation-tolerant non-volatile ternary content addressable memory with magnetic tunnel junction". In: *JSTS: Journal of Semiconductor Technology and Science* 17.3 (2017), pp. 458–464.

[109] R. Govindaraj and S. Ghosh. "Design and analysis of STTRAM-based ternary content addressable memory cell". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.4 (2017), pp. 1–22.

[110] M. Kumar, M.-H. Wu, T.-H. Hou, and M. Suri. "CMOS-RRAM Based Non-Volatile Ternary Content Addressable Memory (nvTCAM)". In: *IEEE Transactions on Nanotechnology* 23 (2024), pp. 203–207.

[111] A. Grossi, E. Vianello, C. Zambelli, P. Royer, J.-P. Noel, B. Giraud, L. Perniola, P. Olivo, and E. Nowak. "Experimental investigation of 4-kb RRAM arrays programming conditions suitable for TCAM". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.12 (2018), pp. 2599–2607.

[112] M.-F. Chang, C.-C. Lin, A. Lee, Y.-N. Chiang, C.-C. Kuo, G.-H. Yang, H.-J. Tsai, T.-F. Chen, and S.-S. Sheu. "A 3T1R nonvolatile TCAM using MLC ReRAM for frequent-off instant-on filters in IoT and big-data processing". In: *IEEE Journal of Solid-State Circuits* 52.6 (2017), pp. 1664–1679.

[113] R. Sharma, N. S. Dhakad, G. S. Reddy, V. Sharma, and S. K. Vishvakarma. "ReCAM: Resistive RAM Digital Content Addressable Memory Using Novel 3T1R Bitcell". In: *2024 8th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. 2024, pp. 1–3.

[114] L.-Y. Huang, M.-F. Chang, C.-H. Chuang, C.-C. Kuo, C.-F. Chen, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, K.-L. Su, et al. "ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing". In: *2014 Symposium on VLSI Circuits Digest of Technical Papers*. IEEE. 2014, pp. 1–2.

[115] C.-C. Lin, J.-Y. Hung, W.-Z. Lin, C.-P. Lo, Y.-N. Chiang, H.-J. Tsai, G.-H. Yang, Y.-C. King, C. J. Lin, T.-F. Chen, et al. "7.4 A 256b-wordlength ReRAM-based TCAM with 1ns search-time and 14× improvement in wordlength-energyefficiency-density product using 2.5 T1R cell". In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. 2016, pp. 136–137.

[116] M.-F. Chang, C.-H. Chuang, Y.-N. Chiang, S.-S. Sheu, C.-C. Kuo, H.-Y. Cheng, J. Sampson, and M. J. Irwin. "Designs of emerging memory based non-volatile TCAM for Internet-of-Things (IoT) and big-data processing: A 5T2R universal cell". In: *2016 IEEE international symposium on circuits and systems (ISCAS)*. IEEE. 2016, pp. 1142–1145.

[117] M.-F. Chang, C.-C. Lin, A. Lee, C.-C. Kuo, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, P.-L. Tseng, H.-Y. Lee, et al. "17.5 A 3T1R nonvolatile TCAM using MLC ReRAM with sub-1ns search time". In: *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE. 2015, pp. 1–3.

[118] P. Junsangsri, J. Han, and F. Lombardi. "Design and comparative evaluation of a PCM-based CAM (content addressable memory) cell". In: *IEEE Transactions on Nanotechnology* 16.2 (2017), pp. 359–363.

[119] J. Li, R. K. Montoye, M. Ishii, and L. Chang. "1 Mb 0.41 μm² 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing". In: *IEEE Journal of Solid-State Circuits* 49.4 (2014), pp. 896–907.

[120] M. Tarkov, F. Tikhonenko, V. Popov, V. Antonov, A. Miakonkikh, and K. Rudenko. "Ferroelectric devices for content-addressable memory". In: *Nanomaterials* 12.24 (2022), p. 4488.

[121] D. R. B. Ly, B. Giraud, J.-P. Noel, A. Grossi, N. Castellani, G. Sassine, J.-F. Nodin, G. Molas, C. Fenouillet-Beranger, G. Indiveri, E. Nowak, and E. Vianello. "In-depth Characterization of Resistive Memory-Based Ternary Content Addressable Memories". In: *2018 IEEE International Electron Devices Meeting (IEDM)*. 2018, pp. 20.3.1–20.3.4.

[122] S. B. Mamaghani, P. Pal, and M. B. Tahoori. "A Dynamic Testing Scheme for Resistive-Based Computation-In-Memory Architectures". In: *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2024, pp. 509–514.

[123] S. H. Moon, P. Dutta, P. Khorrami, S. Bhanja, and D. Reis. "The Impact of Device Technologies on the Design of Non-Volatile Content Addressable Memories". In: *2024 IEEE 24th International Conference on Nanotechnology (NANO)*. IEEE. 2024, pp. 513–516.

[124] M. Si, H.-Y. Cheng, T. Ando, G. Hu, and P. D. Ye. "Overview and outlook of emerging non-volatile memories". In: *Mrs Bulletin* 46.10 (2021), pp. 946–958.

[125] G. Sun, J. Zhao, M. Poremba, C. Xu, and Y. Xie. "Memory that never forgets: Emerging nonvolatile memory and the implication for architecture design". In: *National Science Review* 5.4 (2018), pp. 577–592.

[126] S. Kargar and F. Nawab. "Challenges and future directions for energy, latency, and lifetime improvements in NVMs". In: *Distributed and Parallel Databases* 41.3 (2023), pp. 163–189.

[127] J. J. Sun, M. DeHerrera, B. Hughes, S. Ikegawa, H. K. Lee, F. B. Mancoff, K. Nagel, G. Shimon, S. M. Alam, D. Houssameddine, and S. Aggarwal. "Commercialization of 1Gb Standalone Spin-Transfer Torque MRAM". In: *2021 IEEE International Memory Workshop (IMW)*. 2021, pp. 1–4.

[128] P.-H. Lee, C.-F. Lee, Y.-C. Shih, H.-J. Lin, Y.-A. Chang, C.-H. Lu, Y.-L. Chen, C.-P. Lo, C.-C. Chen, C.-H. Kuo, et al. "33.1 A 16nm 32Mb embedded STT-MRAM with a 6ns read-access time, a 1M-cycle write endurance, 20-year retention at 150° C and MTJ-OTP solutions for magnetic immunity". In: *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. 2023, pp. 494–496.

[129] T. Ogawa, K. Matsubara, Y. Taito, T. Saito, M. Izuna, K. Takeda, Y. Kaneda, T. Shimoi, H. Mitani, T. Ito, and T. Kono. "15.8 A 22nm 10.8Mb Embedded STT-MRAM Macro Achieving over 200MHz Random-Read Access and a 10.4MB/s Write Throughput with an In-Field Programmable 0.3Mb MTJ-OTP for High-End MCUs". In: *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 67. 2024, pp. 290–292.

[130] K.-F. Lin, H. Noguchi, Y.-C. Shih, P.-F. Yuh, Y.-J. Lee, T.-C. Chang, S.-P. Huang, Y.-F. Lin, C.-Y. Lee, Y.-H. Huang, J.-C. Tsai, S. Adham, P. Noel, R. Yazdi, M. Gershoig, Y. Shin, V. Joshi, T. Wong, M.-R. Jiang, J. J. Wu, C.-T. Cheng, Y.-J. Wang, H. Chuang, Y.-D. Chih, Y. Wang, and T.-Y. J. Chang. "15.9 A 16nm 16Mb Embedded STT-MRAM with a 20ns Write Time, a 1012 Write Endurance and Integrated Margin-Expansion Schemes". In: *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 67. 2024, pp. 292–294.

[131] A. Chen. "A review of emerging non-volatile memory (NVM) technologies and applications". In: *Solid-State Electronics* 125 (2016), pp. 25–38.

[132] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao. "Emerging NVM: A survey on architectural integration and research challenges". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23.2 (2017), pp. 1–32.

[133] C. Dou, W.-H. Chen, Y.-J. Chen, H.-T. Lin, W.-Y. Lin, M.-S. Ho, and M.-F. Chang. "Challenges of emerging memory and memristor based circuits: Nonvolatile logics, IoT security, deep learning and neuromorphic computing". In: *2017 IEEE 12th international conference on ASIC (ASICON)*. IEEE. 2017, pp. 140–143.

[134] G. Verma, A. Prajapati, N. Bindal, and B. K. Kaushik. "Comparative analysis of spin based memories for neuromorphic computing". In: *Spintronics XIII*. Vol. 11470. SPIE. 2020, pp. 40–49.

[135] S. C. Krishnan, R. Panigrahy, and S. Parthasarathy. "Error-correcting codes for ternary content addressable memories". In: *IEEE Transactions on Computers* 58.2 (2008), pp. 275–279.

[136] M. A. Alam, M. M. Hussain, and M. S. Hossain. "Variability Analysis of Resistive Ternary Content Addressable Memories". In: *IEEE Transactions on Electron Devices* (2021).

[137] P.-P. Manea, C. Sudarshan, F. Cüppers, and J. P. Strachan. "Non-idealities and design solutions for analog memristor-based content-addressable memories". In: *Proceedings of the 18th ACM International Symposium on Nanoscale Architectures*. 2023, pp. 1–6.

[138] N. H. Weste and D. Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[139] T. Na, S. H. Kang, and S.-O. Jung. "STT-MRAM sensing: a review". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.1 (2020), pp. 12–18.

[140] F. Bernard-Granger, B. Dieny, R. Fascio, and K. 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.

[141] C. Bengel, A. Siemon, F. Cüppers, S. Hoffmann-Eifert, A. Hardtdegen, M. von Witzleben, L. Hellmich, R. Waser, and S. Menzel. "Variability-Aware Modeling of Filamentary Oxide-Based Bipolar Resistive Switching Cells Using SPICE Level Compact Models". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.12 (2020), pp. 4618–4630.

[142] S. Wiefels, C. Bengel, N. Kopperberg, K. Zhang, R. Waser, and S. Menzel. "HRS Instability in Oxide-Based Bipolar Resistive Switching Cells". In: *IEEE Transactions on Electron Devices* 67.10 (2020), pp. 4208–4215.

[143] M. Le Gallo and A. Sebastian. "An overview of phase-change memory device physics". In: *Journal of Physics D: Applied Physics* 53.21 (2020), p. 213002.

[144] M. Le Gallo, A. Athmanathan, D. Krebs, and A. Sebastian. "Evidence for thermally assisted threshold switching behavior in nanoscale phase-change memory cells". In: *Journal of Applied Physics* 119.2 (2016), p. 025704.

[145] H.-S. P. Wong and S. Salahuddin. "Memory leads the way to better computing". In: *Nature Nanotechnology* 10.3 (2015), pp. 191–194.

[146] L. O. Chua. "Resistance switching memories are memristors". In: *Applied Physics A* 102.4 (2011), pp. 765–783.

[147] R. Waser and M. Aono. "Nanoionics-based resistive switching memories". In: *Nature Materials* 6.11 (2007), pp. 833–840.

[148] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Amorim, P. M. L. Gallo, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang. "Neuromorphic computing using non-volatile memory". In: *Advanced Physics: X* 2.1 (2017), pp. 89–124.

[149] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.10 (2014), pp. 2054–2066.

[150] I. Vourkas and G. C. Sirakoulis. "Emerging memristor-based logic circuit design approaches: A review". In: *IEEE Circuits and Systems Magazine* 16.3 (2016), pp. 15–30.

[151] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams. "Memristive switches enable stateful logic operations via material implication". In: *Nature* 464.7290 (2010), pp. 873–876.

[152] L. Xie, H. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui. "Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing". In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2017, pp. 176–181.

[153] T. Liang, X. Li, Z. Li, H. Li, G. Chen, Y. Chen, and H. Lu. "An area-efficient memristor-based trimming circuit for current reference with temperature coefficient optimization capability". In: *IEICE Electronics Express* 22.12 (2025), pp. 20250210–20250210.

[154] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. El Hajjam, R. Crochemore, J.-F. Nodin, P. Olivo, et al. "Fundamental variability limits of filament-based RRAM". In: *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2016, pp. 4–7.

[155] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. "Statistical fault injection: Quantified error and confidence". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, pp. 502–506.

[156] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. S. B. Altaf, N. Vaish, M. D. Hill, and D. A. Wood. "The gem5 simulator". In: *SIGARCH Comput. Archit. News* 39.2 (2011), pp. 1–7.

[157] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrill'on, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, M. Fariborz, A. F. Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, T. M"uck, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. S. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and '. F. Zulian. "The gem5 Simulator: Version 20.0+". In: *CoRR* abs/2007.03152 (2020). arXiv: `2007.03152`.

[158] *gem5 GitHub Repository*. `https://github.com/gem5/gem5`. Accessed: 2023-07-25.

[159] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures". In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2009, pp. 469–480.

[160] X. Dong, C. Xu, Y. Xie, and N. 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.

[161] A. BanaGozar, K. Vadivel, S. Stuijk, H. Corporaal, S. Wong, M. A. Lebdeh, J. Yu, and S. Hamdioui. "CIM-SIM: Computation In Memory SIMuIator". In: *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*. SCOPES '19. Association for Computing Machinery, 2019, pp. 1–4.

[162] L. Xia, B. Li, T. Tang, P. Gu, X. Yin, W. Huangfu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang. "MNSIM: Simulation platform for memristor-based neuromorphic computing system". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016, pp. 469–474.

[163] P.-Y. Chen, X. Peng, and S. Yu. "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.12 (2018), pp. 3067–3080.

[164] S. Mosanu, M. N. Sakib, T. Tracy, E. Cukurtas, A. Ahmed, P. Ivanov, S. Khan, K. Skadron, and M. Stan. "PiMulator: a Fast and Flexible Processing-in-Memory Emulation Platform". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 1473–1478.

[165] C. Yu, S. Liu, and S. Khan. "MultiPIM: A Detailed and Configurable Multi-Stack Processing-In-Memory Simulator". In: *IEEE Computer Architecture Letters* 20.1 (2021), pp. 54–57.

[166] P. C. Santos, B. E. Forlin, and L. Carro. "Sim2PIM: A Fast Method for Simulating Host Independent & PIM Agnostic Designs". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021, pp. 226–231.

[167] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li. "PIMSim: A Flexible and Detailed Processing-in-Memory Simulator". In: *IEEE Computer Architecture Letters* 18.1 (2019), pp. 6–9.

[168] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra. "Tolerating Soft Errors in Processor Cores Using CLEAR (Cross-Layer Exploration for Architecting Resilience)". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.9 (2018), pp. 1839–1852.

[169] E. Cheng, Daniel-Mueller-Gritschneder, J. Abraham, P. Bose, A. Buyuktosunoglu, D. Chen, H. Cho, Y. Li, U. Sharif, K. Skadron, M. Stan, U. Schlichtmann, and S. Mitra. "Cross-Layer Resilience: Challenges, Insights, and the Road Ahead". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Association for Computing Machinery, 2019.

[170] G. Papadimitriou and D. Gizopoulos. "Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA '21)*. 2021, pp. 902–915.

[171] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor". In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, pp. 29–.

[172] G. Papadimitriou and D. Gizopoulos. "AVGI: Microarchitecture-Driven, Fast and Accurate Vulnerability Assessment". In: *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2023, pp. 935–948.

[173] M. Mayahinia, C. Münch, and M. B. Tahoori. "Analyzing and Mitigating Sensing Failures in Spintronic-based Computing in Memory". In: *2021 IEEE International Test Conference (ITC)*. 2021, pp. 268–277.

[174] M. Mayahinia, A. Jafari, and M. B. Tahoori. “Voltage Tuning for Reliable Computation in Emerging Resistive Memories”. In: *2022 IEEE 40th VLSI Test Symposium (VTS)*. 2022, pp. 1–7.

[175] M. E. Fouda, S. Lee, J. Lee, G. H. Kim, F. Kurdahi, and A. M. Eltawi. “IR-QNN Framework: An IR Drop-Aware Offline Training of Quantized Crossbar Arrays”. In: *IEEE Access* 8 (2020), pp. 228392–228408.

[176] S. Lee, G. Jung, M. E. Fouda, J. Lee, A. Eltawil, and F. Kurdahi. “Learning to Predict IR Drop with Effective Training for ReRAM-based Neural Network Hardware”. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6.

[177] Y.-H. Chiang, C.-E. Ni, Y. Sung, T.-H. Hou, T.-S. Chang, and S.-J. Jou. “Hardware-Robust In-RRAM-Computing for Object Detection”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12.2 (2022), pp. 547–556.

[178] M. H. Amin, M. Elbtity, and R. Zand. “Interconnect Parasitics and Partitioning in Fully-Analog In-Memory Computing Architectures”. In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2022, pp. 389–393.

[179] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach. “Transient fault models and AVF estimation revisited”. In: *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. 2010, pp. 477–486.

[180] M. T. Yourst. “PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator”. In: *2007 IEEE International Symposium on Performance Analysis of Systems & Software*. 2007, pp. 23–34.

[181] G. Yalcin, O. S. Unsal, A. Cristal, and M. Valero. “FIMSIM: A fault injection infrastructure for microarchitectural simulators”. In: *2011 IEEE 29th International Conference on Computer Design (ICCD)*. 2011, pp. 431–432.

[182] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos. “Differential Fault Injection on Microarchitectural Simulators”. In: *2015 IEEE International Symposium on Workload Characterization*. 2015, pp. 172–182.

[183] L. Palazzi, G. Li, B. Fang, and K. Pattabiraman. “A Tale of Two Injectors: End-to-End Comparison of IR-Level and Assembly-Level Fault Injection”. In: *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. 2019, pp. 151–162.

[184] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. “gem5-Approxilyzer: An Open-Source Tool for Application-Level Soft Error Analysis”. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, pp. 214–221.

[185] Z. Wan, T. Wang, Y. Zhou, S. S. Iyer, and V. P. Roychowdhury. “Accuracy and Resiliency of Analog Compute-in-Memory Inference Engines”. In: *J. Emerg. Technol. Comput. Syst.* 18.2 (2022).

[186] B. Yan, M. Liu, Y. Chen, K. Chakrabarty, and H. Li. “On Designing Efficient and Reliable Nonvolatile Memory-Based Computing-In-Memory Accelerators”. In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019, pp. 14.5.1–14.5.4.

[187] H. Liu, Z. Qian, W. Wu, H. Ren, Z. Liu, and L. Ni. *AFPR-CIM: An Analog-Domain Floating-Point RRAM-based Compute-In-Memory Architecture with Dynamic Range Adaptive FP-ADC*. 2024. arXiv: 2402.13798 [eess.SY].

[188] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear. *The art of multiprocessor programming*. Newnes, 2020.

[189] A. Peleg and U. Weiser. "MMX technology extension to the Intel architecture". In: *IEEE Micro* 16.4 (1996), pp. 42–50.

[190] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech. "Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments". In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, pp. 26–38.

[191] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech. "Soft Error Effects on Arm Microprocessors: Early Estimations versus Chip Measurements". In: *IEEE Transactions on Computers* 71.10 (2022), pp. 2358–2369.

[192] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra. "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design". In: *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*. 2013.

[193] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. "CACTI 6.0: A tool to model large caches". In: *HP laboratories* 27 (2009), p. 28.

[194] N. I. O. STANDARDS and T. G. MD. "Announcing the Advanced Encryption Standard (AES)". In: *U.S. Department of Commerce* (2001).

[195] A. Pappalardo, G. Franco, nickfraser, I. Colbert, P. M. Lago, F. Grob, T. Costigan, O. Savolainen, A. Stoian, jinchen, A. Gerdelan, alexredd99, F. Jentzsch, J. Duarte, L. Montero, MichalMachura, O. Peracha, S. Khan, K. L. Witham, T. Paine, Y. Umuroglu, derpda, hkayann, vfdev, and rgb000000. *Xilinx/brevitas: Release v0.12.1*. Version v0.12.1. 2025.

[196] L. Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[197] J. Zhang and E. Sadredini. "Inhale: Enabling high-performance and energy-efficient in-SRAM cryptographic hash for IoT". In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 2022, pp. 1–9.

[198] J. Zhang, H. Naghibijouybari, and E. Sadredini. "Sealer: In-SRAM AES for High-Performance and Low-Overhead Memory Encryption". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 2022, pp. 1–6.

[199] B. Narasimham, V. Chaudhary, M. Smith, L. Tsau, D. Ball, and B. Bhuva. "Scaling Trends in the Soft Error Rate of SRAMs from Planar to 5-nm FinFET". In: *2021 IEEE International Reliability Physics Symposium (IRPS)*. 2021, pp. 1–5.

[200] G. Papadimitriou and D. Gizopoulos. “Characterizing Soft Error Vulnerability of CPUs Across Compiler Optimizations and Microarchitectures”. In: *2021 IEEE International Symposium on Workload Characterization (IISWC)*. 2021, pp. 113–124.

[201] D. M. Brooks et al. “Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors”. In: *IEEE Micro* 20.6 (2000), pp. 26–44.

[202] A. Fantini, L. Goux, R. Degraeve, D. Wouters, N. Raghavan, G. Kar, A. Belmonte, Y.-Y. Chen, B. Govoreanu, and M. Jurczak. “Intrinsic switching variability in HfO 2 RRAM”. In: *2013 5th IEEE International Memory Workshop*. IEEE. 2013, pp. 30–33.

[203] J.-Y. Sim, H. Yoon, K.-C. Chun, H.-S. Lee, S.-P. Hong, K.-C. Lee, J.-H. Yoo, D.-I. Seo, and S.-I. o. “A 1.8-V 128-Mb mobile DRAM with double boosting pump, hybrid current sense amplifier, and dual-referenced adjustment scheme for temperature nsor”. In: *IEEE Journal of Solid-State Circuits* 38.4 (2003), pp. 631–640.

[204] Y. Zhao, H. Chen, P. Liu, J. Wu, and D. Luo. “An improved reconfigurable logic in resistive random access memory”. In: *Integration* 87 (2022), pp. 169–175.

[205] T. Zanotti, P. Pavan, and F. M. Puglisi. “Multi-input logic-in-memory for ultra-low power non-von Neumann computing”. In: *Micromachines* 12.10 (2021), p. 1243.

[206] Y. Wang, D. Song, J. Dai, E. Deng, Y. Gong, and W. Liu. “Dual-Swing-Sample-and-Couple Sense Amplifier With Large Sensing Margin for STT-MRAM”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2025).

[207] J. Kim, Y. Jang, T. Kim, and J. Park. “A dual-domain dynamic reference sensing for reliable read operation in SOT-MRAM”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 69.5 (2022), pp. 2049–2059.

[208] J. Yan, D.-S. Liu, H.-M. Yu, H. Meng, and P. Liu. “A high-reliability reading scheme for STT-MRAM with a symmetric dual-reference sensing amplifier”. In: *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE. 2018, pp. 1–3.

[209] B.-K. An, X. Zhang, A. T. Do, and T. T.-H. Kim. “Design of a Reliable Current Sense Amplifier with Dynamic Reference for Resistive Memory”. In: *JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE* 24.3 (2024), pp. 226–239.

[210] B.-K. An, X. Zhang, A. T. Do, and T. T.-H. Kim. “Design of a current sense amplifier with dynamic reference for reliable resistive memory”. In: *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE. 2023, pp. 1–5.

[211] J. Yu, H. A. Du Nguyen, M. A. Lebdeh, M. Taouil, and S. Hamdioui. “Enhanced scouting logic: A robust memristive logic design scheme”. In: *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE. 2019, pp. 1–6.

[212] F. Liu, S. Zhang, and X. Cui. “The design method of logic circuits based on the voltage-input enhanced scouting logic gates”. In: *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2022, pp. 136–142.

[213] T. Zanotti. "Reliability and Prospects of Logic-in-Memory Circuits". In: *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE. 2022, pp. 369–371.

[214] M. Fieback, C. Münch, A. Gebregiorgis, G. C. Medeiros, M. Taouil, i. Hamdioui, and M. Tahoori. "PVT Analysis for RRAM and STT-MRAM-based Logic Computation-in-Memory". In: *2022 IEEE European Test Symposium (ETS)*. IEEE. 2022, pp. 1–6.

[215] F. Pinto and I. Vourkas. "Robust circuit and system design for general-purpose computational resistive memories". In: *Electronics* 10.9 (2021), p. 1074.

[216] F. Pinto and I. Vourkas. "Design Considerations for the Development of Computational Resistive Memories". In: *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*. IEEE. 2021, pp. 1–4.

[217] L. Brackmann, T. Ziegler, D. J. Wouters, and S. Menzel. "Experimental verification and evaluation of non-stateful logic gates in resistive RAM". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2024).

[218] S. Hamdioui, M. Fieback, S. Nagarajan, and M. Taouil. "Testing computation-in-memory architectures based on emerging memories". In: *2019 IEEE International Test Conference (ITC)*. IEEE. 2019, pp. 1–10.

[219] L. Yao, J. Wu, P. Liu, and S.-K. Lam. "DAGSIS: A DAG-Aware MAGIC based Synthesis Framework for In-Memory Computing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025).

[220] O. Chatzopoulos, G. Papadimitriou, V. Karakostas, and D. Gizopoulos. "Gem5-marvel: Microarchitecture-level resilience analysis of heterogeneous soc architectures". In: *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2024, pp. 543–559.

[221] A. Genz and F. Bretz. *Computation of multivariate normal and t probabilities*. Springer Science & Business Media, 2009.

[222] M. Poremba, T. Zhang, and Y. Xie. "NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems". In: *IEEE Comput. Archit. Lett.* 14.2 (2015), pp. 140–143.

[223] Everspin Technologies, Inc. *EMD4E001GAS2 1Gb Non-Volatile ST-DDR4 Spin-transfer Torque MRAM Datasheet*. Technical datasheet. Revision 1.3. 2022.

[224] Y. Wang, H. Cai, L. A. d. B. Naviner, Y. Zhang, X. Zhao, E. Deng, J.-O. Klein, and W. Zhao. "Compact Model of Dielectric Breakdown in Spin-Transfer Torque Magnetic Tunnel Junction". In: *IEEE Transactions on Electron Devices* 63.4 (2016), pp. 1762–1767.

[225] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009.

[226] R. Sedgewick and K. Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011, pp. I–XII, 1–955.

[227] H. Plattner. *A course in in-memory data management*. Springer, 2013.

[228] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. "Accelerating large-scale inference with anisotropic vector quantization". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020.

[229] S. P. R. Maharshi J. Pathak Ronit L. Patel. "Comparative Analysis of Search Algorithms". In: *International Journal of Computer Applications* 179.50 (2018), pp. 40–43.

[230] N. Sultana, S. Paira, S. Chandra, and S. K. S. Alam. "A brief study and analysis of different searching algorithms". In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. 2017, pp. 1–4.

[231] C. Balkesen, G. Alonso, J. Teubner, and M. T. Özsu. "Multi-core, main-memory joins: sort vs. hash revisited". In: *Proc. VLDB Endow.* 7.1 (2013), pp. 85–96.

[232] G. Blelloch. "Scans as primitive parallel operations". In: *IEEE Transactions on Computers* 38.11 (1989), pp. 1526–1538.

[233] A. Gangal, P. Kumar, and S. Kumari. *Complete Scanning Application Using OpenCv*. 2021. arXiv: 2107.03700 [cs.CV].

[234] N. J. Lisa, A. Ungethüm, D. Habich, N. D. A. Tuan, A. Kumar, and W. Lehner. "Column Scan Optimization by Increasing Intra-Instruction Parallelism". In: *Proceedings of the 7th International Conference on Data Science, Technology and Applications*. SCITEPRESS - Science and Technology Publications, Lda, 2018, pp. 344–353.

[235] G. Estrin and R. H. Fuller. "Some applications for content-addressable memories". In: *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference*. AFIPS '63 (Fall). Association for Computing Machinery, 1963, pp. 495–508.

[236] T. Kohonen. *Content-Addressable Memories*. Springer Berlin Heidelberg, 1980.

[237] N. Gupta, A. Makosiej, A. Amara, A. Vladimirescu, and C. Anghel. "Content-Addressable Memories". In: *TFET Integrated Circuits: From Perspective Towards Reality*. Springer International Publishing, 2021, pp. 73–103.

[238] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey. "Exploring hyperdimensional associative memory". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2017, pp. 445–456.

[239] P. R. Genssler, M. Mayahinia, S. Thomann, M. Tahoori, and H. Amrouch. "DropHD: Technology/Algorithm Co-design for Reliable Energy-efficient NVM-based Hyperdimensional Computing under Voltage Scaling". In: *IEEE DATE*. 2024.

[240] S. Thomann, H. L. Nguyen, P. R. Genssler, and H. Amrouch. "All-in-memory brain-inspired computing using fefet synapses". In: *Frontiers in Electronics* 3 (2022), p. 833260.

[241] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müller, S. Beyer, M. Niemier, X. S. Hu, et al. "Ferroelectric ternary content-addressable memory for one-shot learning". In: *Nature Electronics* 2.11 (2019), pp. 521–529.

[242] H. E. Barkam, S. Yun, H. Chen, P. Gensler, A. Mema, A. Ding, G. Michelogiannakis, H. Amrouch, and M. Imani. "Reliable hyperdimensional reasoning on unreliable emerging technologies". In: *2023 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2023, pp. 1–9.

[243] M. Zhang, Y. Zhu, C.-H. Chan, and R. P. Martins. "An 8-bit 10-GS/s 16× interpolation-based time-domain ADC with< 1.5-ps uncalibrated quantization steps". In: *IEEE Journal of Solid-State Circuits* 55.12 (2020), pp. 3225–3235.

[244] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. "BLAST+: architecture and applications". In: *BMC bioinformatics* 10.1 (2009), p. 421.

[245] A. Thomas, S. Dasgupta, and T. Rosing. "A Theoretical Perspective on Hyperdimensional Computing". In: *Journal of Artificial Intelligence Research* 72 (2021), pp. 215–249.

[246] M. Hamadouche, K. Zebbiche, M. Guerroumi, H. Tebbi, and Y. Zafoune. "A comparative study of perceptual hashing algorithms: Application on fingerprint images". In: *The 2nd International Conference on Computer Science's Complex Systems and their Applications*. 2021.

[247] S. Li, L. Liu, P. Gu, C. Xu, and Y. Xie. "NVSim-CAM: A Circuit-Level Simulator for Emerging Nonvolatile Memory based Content-Addressable Memory". In: *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016, pp. 1–7.

[248] L. Liu, M. M. Sharifi, R. Rajaei, A. Kazemi, K. Ni, X. Yin, M. Niemier, and X. S. Hu. "Eva-CAM: A Circuit/Architecture-Level Evaluation Tool for General Content Addressable Memories". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022.

[249] Tamber. *Steam Video Games*. `https://www.kaggle.com/datasets/tamber/steam-video-games`. Accessed: 2025-07-02. 2017.

[250] E. Loghmani and M. Fazli. "Effect of Choosing Loss Function when Using T-batching for Representation Learning on Dynamic Networks". In: *CoRR* abs/2308.06862 (2023). arXiv: `2308.06862`.

[251] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`. University of Toronto, 2009.

[252] Human Genome Project. *Human Genome Project, Rough Draft, Chromosome Number 21*. Project Gutenberg. 2000.

[253] Fujitsu Semiconductor Memory Solution Limited. *MB85AS8MT Memory ReRAM 8M (1024K x 8) SPI Datasheet*. Technical datasheet. Revision 2.0. 2021.