Karlsruher Institut für Technologie

Methods for Security and Functional Safety in Reconfigurable AI Accelerators

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Vincent Meyers

aus Herne, Deutschland

Tag der mündlichen Prüfung: 10.06.2026

1. Referent: Prof. Dr. Mehdi Baradaran Tahoori
Karlsruher Institut für Technologie

2. Referent: Prof. Dr.-Ing. Ahmad-Reza Sadeghi
Technische Universität Darmstadt

Acknowledgments

I would like to express my sincere gratitude to all those who supported me throughout my doctoral studies and contributed to the completion of this thesis.

First and foremost, I would like to thank my supervisor, Prof. Dr. Mehdi B. Tahoori, for his continuous guidance and support. His insights, encouragement, and trust have been invaluable throughout this work, from the early research ideas to the final stages of this dissertation.

I am especially grateful to Dr.-Ing. Dennis Gnad, who supervised my first research work during my Master's thesis and sparked my interest in pursuing a PhD. Our continued collaboration over the years, resulting in numerous joint publications, has greatly shaped my research and academic development.

I would also like to thank Dr.-Ing. Michael Hefenbrock for the many fruitful collaborations and joint publications. His expertise and discussions have significantly contributed to this work.

I also want to thank my former students Johannes, Henrik, Daniel, Simon and Ahmet for their excellent work and contributions. Supervising you has been a rewarding experience, and your motivation and ideas have positively influenced this research.

Furthermore, I thank all my colleagues at the institute for the supportive and inspiring working environment. The discussions, collaborations, and shared experiences—both academic and beyond—made this time particularly enjoyable and enriching.

Finally, I am deeply grateful to my friends and family for their unwavering support. Especially, I want to thank Maha, Daniel, Laura, Dani, Maika and Alina for their constant support, encouragement, and for always being there for me. Their patience, understanding, and constant motivation have been essential throughout this journey.

Vincent Meyers

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

Karlsruhe, 10. Juni 2026
Vincent Meyers

Abstract

Neural network accelerators implemented on field-programmable gate arrays (FPGAs) enable low-latency and energy-efficient inference in cloud, edge, and embedded systems. At the same time, their deployment in shared and potentially adversarial environments exposes them to physical side-channel leakage, especially through data-dependent power consumption. This thesis investigates such leakage in realistic FPGA-based neural network accelerators and shows that it has a dual role: it poses security risks, but also provides valuable observability for monitoring and functional safety.

First, the thesis demonstrates practical side-channel attacks against realistic accelerator implementations. It shows that architectural properties such as folding can be reverse-engineered from power traces and that private inputs, including images and audio signals, can be reconstructed directly from on-chip voltage fluctuations using generative models. These attacks remain effective across varying operating conditions and hardware instances.

Second, the thesis shows that side-channel leakage can be amplified and manipulated during training. By incorporating hardware-aware objectives, the correlation between computation and observable power consumption can be strengthened, enabling passive recovery of output labels without modifying the hardware. The work further explores malicious weight manipulation in federated learning, showing that crafted updates can increase power consumption and degrade efficiency while largely preserving functional accuracy.

Third, the thesis investigates defensive uses of side-channel information. It introduces side-channel-aware training methods to reduce leakage and a power-based fingerprinting technique for remote model identification. Finally, it demonstrates that on-chip voltage sensors can be used for concurrent detection of out-of-distribution inputs and hardware faults during inference with minimal overhead.

Overall, this work establishes side-channel leakage as a fundamental property of reconfigurable AI accelerators that must be considered systematically. While it threatens confidentiality and integrity, it can also be exploited as a monitoring interface to improve the security, robustness, and functional safety of FPGA-based AI systems.

Zusammenfassung

Neuronale Netzwerkbeschleuniger, die auf Field-Programmable Gate Arrays (FPGAs) implementiert werden, ermöglichen latenzarme und energieeffiziente Inferenz in Cloud-, Edge- und eingebetteten Systemen. Gleichzeitig macht ihre Bereitstellung in gemeinsam genutzten und potenziell adversariellen Umgebungen sie anfällig für physikalische Seitenkanäle, insbesondere durch datenabhängigen Energieverbrauch. Diese Dissertation untersucht solche Seitenkanäle in realistischen FPGA-basierten Beschleunigern für neuronale Netzwerke und zeigt, dass ihnen eine doppelte Rolle zukommt: Sie stellen ein Sicherheitsrisiko dar, bieten zugleich jedoch wertvolle Beobachtbarkeit für Monitoring und funktionale Sicherheit.

Zunächst demonstriert die Arbeit praktische Seitenkanalangriffe auf realistische Beschleunigerimplementierungen. Es wird gezeigt, dass architektonische Eigenschaften wie Folding aus Leistungsmessungen rückentwickelt werden können und dass private Eingaben, einschließlich Bild- und Audiosignalen, mithilfe generativer Modelle direkt aus integrierten Spannungsschwankungen rekonstruiert werden können. Diese Angriffe bleiben auch unter variierenden Betriebsbedingungen und über verschiedene Hardwareinstanzen hinweg wirksam.

Darüber hinaus zeigt die Dissertation, dass sich Seitenkanal-Informationen bereits während des Trainings gezielt verstärken und manipulieren lassen. Durch die Einbeziehung hardwarebewusster Optimierungsziele kann die Korrelation zwischen Berechnung und beobachtbarem Energieverbrauch erhöht werden, sodass eine passive Rekonstruktion von Ausgabelabels ohne Änderungen an der Hardware möglich wird. Weiterhin untersucht die Arbeit bösartige Gewichtsmanipulationen im föderierten Lernen und zeigt, dass gezielt erzeugte Updates den Energieverbrauch erhöhen und die Effizienz verschlechtern können, während die funktionale Genauigkeit weitgehend erhalten bleibt.

Darüber hinaus untersucht die Dissertation defensive Nutzungen von Seitenkanalinformationen. Sie stellt seitenkanalbewusste Trainingsverfahren zur Reduktion von Seitenkanälen sowie eine leistungsbasierte Fingerprinting-Methode zur entfernten Modellidentifikation vor. Abschließend wird gezeigt, dass integrierte Spannungssensoren zur nebenläufigen Erkennung von Out-of-Distribution-Eingaben und Hardwarefehlern während der Inferenz mit minimalem Overhead eingesetzt werden können.

Insgesamt zeigt diese Arbeit, dass Seitenkanäle eine grundlegende Eigenschaft rekonfigurierbarer KI-Beschleuniger darstellen, die systematisch berücksichtigt werden muss. Während sie Vertraulichkeit und Integrität gefährden, können sie zugleich als Monitoring-Schnittstelle genutzt werden, um die Sicherheit, Robustheit und funktionale Sicherheit FPGA-basierter KI-Systeme zu verbessern.

Contents

List of own publications included in this thesis

Transactions & Articles

[1] V. Meyers, D. Gnad, and M. Tahoori, "Active and passive physical attacks on neural network accelerators", *IEEE Design & Test*, 2023

[2] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Leveraging neural trojan side-channels for output exfiltration", *Cryptography*, vol. 9, no. 1, p. 5, 2025

Conferences

[3] V. Meyers, D. Gnad, and M. Tahoori, "Reverse engineering neural network folding with remote fpga power analysis", in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2022, pp. 1–10

[4] L. Huegle, M. Gotthard, V. Meyers, J. Krautter, D. R. Gnad, and M. B. Tahoori, "Power2picture: Using generative cnns for input recovery of neural network accelerators through power side-channels on fpgas", in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2023, pp. 155–161

[5] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Remote identification of neural network fpga accelerators by power fingerprints", in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2023, pp. 259–264

[6] D. Gnad, J. Krautter, A. Kritikakou, V. Meyers, P. Rech, J. E. R. Condia, A. Ruospo, E. Sanchez, F. F. dos Santos, O. Sentieys, et al., "Reliability and security of ai hardware", in *ETS 2024-29th IEEE European Test Symposium*, IEEE, 2024, pp. 1–10

[7] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Trained to leak: Hiding trojan side-channels in neural network weights", in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2024, pp. 122–127

[8] V. Meyers, D. Gnad, and M. Tahoori, "Out-of-distribution detection using power-side channels for improving functional safety of neural network fpga accelerators", in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2024, pp. 1–2

[9] V. Meyers, M. Hefenbrock, M. Sadeghipourrudsari, D. Gnad, and M. Tahoori, "Towards

functional safety of neural network hardware accelerators: Concurrent out-of-distribution detection in hardware using power side-channel analysis", in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1413–1419

[10] M. S. Akram, V. Meyers, M. Tahoori, B. S. Varma, and D. Finlay, "Evoweight: Sponge poisoning of fpga-based dnn accelerators in differential private secure federated learning", in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2025, pp. 182–193

[11] V. Meyers, M. Sadeghipourrudsari, and M. Tahoori, "Concurrent fault detection for binary neural network accelerators via on-chip voltage monitoring", in *2026 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2026

[12] J. Reibold, V. Meyers, and M. Tahoori, "Talking traces: Audio reconstruction power side-channel attack on neural network fpga accelerators", in *2026 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2026, accepted

List of other publications not included in this thesis

[13] V. Meyers, D. R. Gnad, N. M. Dang, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Stealthy logic misuse for power analysis attacks in multi-tenant fpgas (extended version)", *Cryptology ePrint Archive*, 2023

[14] M. S. Roodsari, J. Krautter, V. Meyers, and M. Tahoori, "E 3 hdc: Energy efficient encoding for hyper-dimensional computing on edge devices", in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2024, pp. 274–280

[15] M. S. Akram, B. S. Varma, V. Meyers, M. Tahoori, and D. Finlay, "F2opt: Novel fine-tuning and folding algorithms for fpga-based dnn accelerators", in *2025 35th International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2025

[16] M. S. Roodsari, V. Meyers, and M. Tahoori, "Ced-hdc: Lightweight concurrent error detection for reliable hyperdimensional computing", in *2025 IEEE 43rd VLSI Test Symposium (VTS)*, IEEE, 2025, pp. 1–7

[17] M. S. Roodsari, V. Meyers, and M. Tahoori, "Lightweight concurrent out-of-distribution detection in hyperdimensional computing hardware", in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2025, pp. 1–7

[18] M. Tahoori, V. Meyers, M. Sadeghipour Roodsari, H. Xu, J. Becker, T. Harbaum, F. Frombach, J. Hoefer, G. Sotiropoulos, J. Henkel, et al., "Special sessions-hardware-software co-design for machine learning systems made open-source", in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2025,

pp. 23–32

[19] J. Henkel, M. Tahoori, H. Khdr, H. Nassar, V. Meyers, D. Chen, S. Yildirim, Y. Huang, N. R. Saxena, S. Hukerikar, et al., “Hardware-software co-design for highly optimized, customized, and reliable ai systems”, in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2025, pp. 1–9

[20] H. Xu, S. Meschkov, V. Meyers, and M. Tahoori, “Pwnn: Power-wasting neural network as remote fault injector”, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2026, no. 1, pp. 448–471, 2026

Part I.

Preliminaries

1. Introduction

Neural networks (NNs) are increasingly deployed in safety-critical and privacy-sensitive domains, ranging from medical diagnostics and industrial inspection to autonomous systems and cloud-based analytics [21]. Developing such models requires extensive data curation, repeated architecture exploration, and large-scale training, which makes pre-trained models valuable intellectual property and attractive targets for theft [22]. To reduce deployment costs and improve energy efficiency, many applications rely on FPGA-based accelerators, which offer low-latency inference at significantly reduced power budgets compared to GPUs [23]. At the same time, these platforms are frequently deployed in hostile or multi-tenant environments, where adversaries may exploit shared power-delivery networks to mount remote physical attacks.

Side-channel attacks leverage data-dependent variations in physical behavior to infer confidential information, such as intermediate states, input data, or model parameters [24], [25]. On modern FPGAs, these attacks can be launched remotely, using on-chip sensors such as TDCs or ring oscillators to observe supply-voltage fluctuations without physical contact [26], [27]. Recent studies demonstrated the feasibility of recovering NN architectures, activation patterns, or user inputs from these remote measurements [28], [29], [30]. However, many works rely on simplified accelerator designs and overlook implementation details such as pipelining or neuron and synapse folding, which strongly influence the switching activity and thus the leakage characteristics [3].

As NN accelerators become widely deployed across cloud, edge, and embedded systems, side-channel leakage emerges not only as a security risk but also as a source of observability that can be exploited for runtime monitoring. This thesis investigates side-channel leakage in realistic FPGA-based NN accelerators from both perspectives: as a threat to confidentiality and secure deployment, and as a signal that can be leveraged for concurrent safety monitoring. To this end, we develop practical attacks, lightweight countermeasures, and voltage-monitoring-based methods for detecting OOD inputs and hardware faults during inference.

Figure 1.1 summarizes the central thesis of this work: the same side-channel signals that enable attack can also be exploited for protection and runtime monitoring. Side-channel information surrounds the accelerator as both an attack surface and a monitoring interface, enabling reverse engineering, input recovery, and output exfiltration, while also supporting functions such as concurrent OOD detection and model integrity verification. Motivated by this duality, our contributions span four key research directions, each addressing a critical limitation in existing attacks, defenses, or methods for functional safety.

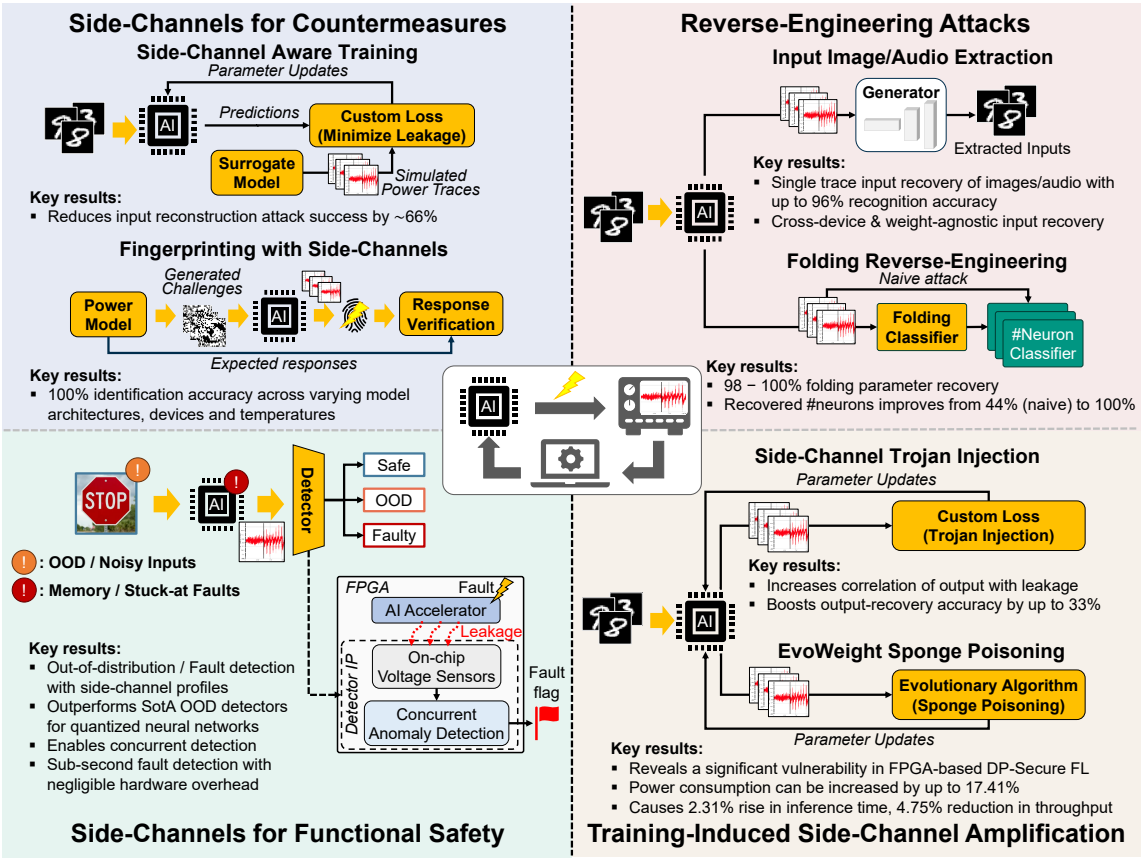


Figure 1.1.: AI accelerator with side-channel measurements in a loop. Side-channels enable reverse engineering, input and output extraction, but the same signals can also support countermeasures and safety mechanisms such as OOD detection and side-channel-aware training.

1.1. Contributions

The contributions of this thesis are centered on the dual role of side-channel information in FPGA-based NN accelerators. On the one hand, data-dependent power fluctuations expose an attack surface that can be exploited for reverse engineering, input recovery, and output exfiltration. On the other hand, the same physical signals provide observability into the accelerator's runtime behavior and can therefore be leveraged for protection, model verification, and functional safety monitoring.

Accordingly, this thesis is organized into four research directions that cover both offensive and defensive uses of side-channel information in realistic, quantized, and resource-constrained FPGA implementations. A broader overview of physical attacks and countermeasures on NN accelerators is provided in [1]. The following sections summarize the main contributions of this thesis in each of these directions.

1.1.1. Reverse Engineering Folding Parameters and Remote Input Recovery with Generative CNNs

NN FPGA accelerators often rely on neuron and synapse folding to fit large NNs into limited hardware resources. This temporal reuse also shapes the power signatures observed by an adversary. We show that existing profiled attacks struggle to infer the original layer widths when folding is ignored, recovering the number of neurons with only 44% accuracy [3]. To overcome this limitation, we utilize time-series analysis that isolates periodic switching behavior induced by folding, enabling accurate reconstruction of the folding factor and the effective layer topology. Once the folding divisor is recovered, specialized classifiers trained per folding configuration can identify the true neuron count with 100% accuracy across all evaluated networks.

Furthermore, we introduce two generative remote side-channel attacks that reconstruct input images and audio from on-chip voltage fluctuations [4], [12]. Our approach trains a generative convolutional model to map voltage traces to corresponding input data, allowing input extraction even under varying temperature conditions, compilation differences, and hardware platforms. In contrast to earlier methods that depend on detailed accelerator knowledge or physical probing, our attack remains effective even when the profiling and victim models differ in their weights and does not require repeated measurements.

1.1.2. Training-Induced Side-Channel Amplification and Manipulation

While prior works focus on recovering inputs or internal states, obtaining the classifier's output from power measurements poses a more challenging problem due to the weak natural correlation between predictions and switching activity. We propose a hardware-aware training method that embeds a stealthy Trojan side-channel directly into the model weights, thereby amplifying the correlation between power consumption and output labels [2], [7]. This enables, for the first time, passive output recovery attacks on FPGA-based accelerators without modifying the hardware or introducing observable overheads.

Additionally, we explore evolutionary poisoning of weights in differentially private federated learning, revealing a new class of stealthy sponge-like attacks that modify power consumption and thermal behavior without degrading accuracy [10].

1.1.3. Side-Channel Aware Training and Power Fingerprinting

To defend against side-channel attacks, we develop a differentiable surrogate model that approximates accelerator-level power consumption and integrate it into the training loop to suppress sensitive leakage [6]. We propose a novel fingerprinting technique for identifying NN accelerators in cloud environments based on their data-dependent power consumption patterns [5]. Using an abstract power model, we synthesize adversarial fingerprinting inputs that maximize inter-model separability in real power measurements.

This allows reliable model identification without requiring access to output labels, unlike existing fingerprinting or watermarking methods that rely on decision-boundary queries or architecture-specific activation functions. Our approach remains effective under chip-to-chip variations and environmental fluctuations, enabling practical model authentication in multi-tenant deployments.

1.1.4. Concurrent Fault and Out-of-Distribution Detection

Finally, we demonstrate how the same side-channel signals exploited by adversaries can be repurposed for functional safety. Out-of-distribution (OOD) data refers to inputs that lie outside the statistical support of the training distribution, such as corrupted sensor readings, unseen environmental conditions, or entirely different object classes, all of which may lead to unsafe or unpredictable model behavior. We show that on-chip voltage sensors enable low-cost, concurrent detection of both OOD inputs and hardware faults during inference. For OOD detection, we learn power profiles of in-distribution samples and apply scoring functions that flag anomalous behavior before the prediction is produced [9].

For fault detection, we leverage subtle voltage deviations caused by bit flips and logic-level errors to train classifiers that operate entirely independently of the accelerator logic, requiring no architectural modifications [11]. This results in a novel gray-box detection mechanism that runs in parallel to the NN accelerator with minimal latency or area overhead.

1.2. Outline

The remainder of this dissertation is organized as follows.

Chapter 2 provides the technical background on FPGA-based neural network accelerators, side-channel measurements, attacks on AI hardware, and functional-safety concepts. Chapter 3 presents reverse-engineering attacks on AI hardware, including the recovery of folding parameters as well as remote image and audio input reconstruction from power side-channels. Chapter 4 investigates training-induced side-channel amplification, including output exfiltration through weight-embedded Trojan side-channels and sponge-poisoning attacks in differential private secure federated learning. Chapter 5 introduces defensive uses of side-channel information through power fingerprinting for model identification and side-channel-aware neural network training for leakage reduction. Chapter 6 shows how side-channel signals can be repurposed for functional safety through concurrent out-of-distribution detection and hardware fault detection using on-chip voltage monitoring. Chapter 7 concludes the thesis and discusses future research directions.

2. Background

The sections in this chapter were partially overtaken from previously published works included in this thesis, which were co-authored with (in no particular order): Dennis Gnad, Michael Hefenbrock, Mahboobe Sadeghipour Roodsari, Mehdi Tahoori.

2.1. Neural Networks

Artificial neural networks (ANNs) are machine learning models inspired by biological neural systems that learn complex relationships between inputs and outputs. They are widely used for tasks such as function approximation, time-series prediction, classification, and pattern recognition. An ANN consists of interconnected nodes called perceptrons, which compute their output as a non-linear function of the weighted sum of their inputs, as illustrated in Figure 2.1. In hardware implementations, the weighted sum is typically realized using adder trees and multipliers or digital signal processors (DSPs) that support matrix-vector or matrix-matrix operations. The connection weights determine the strength of the transmitted signals and are optimized during training using gradient descent or its variants. A special class of neural networks are Binary Neural Networks (BNNs), in which parameters are restricted to $\{-1, 1\}$, significantly reducing memory and computation requirements and making them well suited for embedded systems.

This section is divided into three parts. First, multilayer perceptrons (MLPs) are introduced, followed by a discussion of common activation functions and finally convolutional neural networks (CNNs).

2.1.1. Multilayer Perceptron

Multilayer perceptrons (MLPs) consist of multiple perceptrons arranged in layers, where each neuron receives inputs from the previous layer and forwards its output to the next layer [21]. Figure 2.2 illustrates the layered structure of an MLP. Neurons within the same layer are not interconnected. While a single perceptron can only represent linear functions, stacking multiple layers enables MLPs to approximate complex non-linear functions.

Layers that do not correspond to the final output are referred to as hidden layers. These layers learn intermediate features that are not directly observable in the input or output data [21]. With a sufficient number of hidden units, MLPs can approximate arbitrarily complex functions.

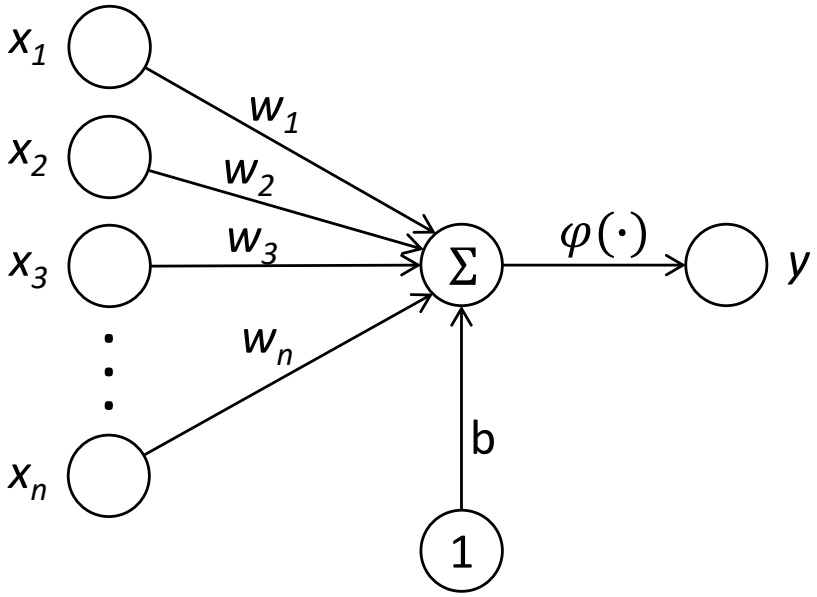


Figure 2.1.: A single perceptron with n inputs x and weights w, a bias with weight b and activation function φ

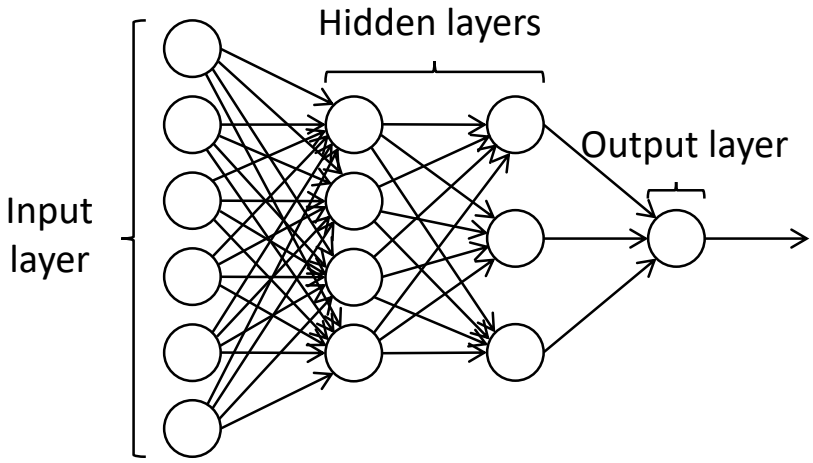


Figure 2.2.: A multilayer perceptron (MLP) with 2 hidden layers

2.1.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) extend MLPs by replacing fully connected operations with convolutional layers in at least part of the network [21]. They are particularly effective for image processing tasks because convolution reduces the dimensionality of input data while preserving spatial information.

In a convolutional layer, a small kernel (or filter) slides over the input and performs a dot product with local input regions, producing feature maps. In addition to common neural network parameters such as the number of layers and neurons, CNNs introduce architectural parameters including filter size, stride, pooling operations, and padding. Only

the weights of the filters are learned during training, while the remaining parameters define the network architecture.

2.1.3. Neural Network Training

NNs can be seen as flexible, parametric function approximators, expressing a function $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^D \rightarrow \mathbb{R}^C$. Here, D is the dimensionality of the inputs $\mathbf{x}$ and C resembles the dimension of the outputs $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$. A classical task of NNs is classification (e.g, of images). In classification, one of C classes should be predicted. Hence, each output of $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = [f(\mathbf{x})_1, \cdots, f(\mathbf{x})_C]$ corresponds to one of C classes. For a given $\mathbf{x}$, the index i relating to the maximum entry of $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$ signifies the intended output. To train an neural network on a given dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n) \mid n = 1, \cdots, N\}$, a loss function Loss $(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_n), y_n)$, measuring the mismatch between $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_n)$ and $y_n \in \{1, \cdots, C\}$, is defined. Common loss functions are, e.g., the cross entropy. Training the neural network then relates to minimizing the loss functions with respect to the parameters $\boldsymbol{\theta}$ using gradient-based optimisation [21].

2.1.4. The FINN Framework

FINN is a open source framework developed by Xilinx for generating quantized neural network inference accelerators to be run on FPGAs [31]. The framework supports the mapping of traditional artifical neural networks, i.e. Multi-Layer Perceptrons (MLPs), and Convolutional Neural Networks (CNNs) with various topologies, while considering FPGA resource limitations and overall throughput. The expected networks for FINN need to be quantized and in the Open Neural Network Exchange (ONNX) representation [32]. This is done by Brevitas, which is a PyTorch library [33] for quantization-aware training with support for exporting FINN-ONNX models.

The accelerator generation is achieved by the following general steps:

1. Train a quantized version of a neural network model using Brevitas [34] in PyTorch
2. Export the model as a QONNX graph [35]
3. Apply streamlining transformations to the model graph to simplify it and reduce the number of operations
4. Convert operations to the corresponding hardware layers
5. Adjust the amount of parallelization for each layer (folding)
6. Synthesize the hardware for a specific FPGA

To guarantee area constraints, FINN can appropriately apply time-multiplexing through *folding*, matching the network layers to available computational units. The units of a hardware layer are called *Matrix–Vector–Threshold Unit* (MVTU), of which the folding can be controlled by the number of physical elements (PE) and the number of SIMD lanes (S), as displayed in Figure 2.3. The PEs are hardware implementations of neurons in a

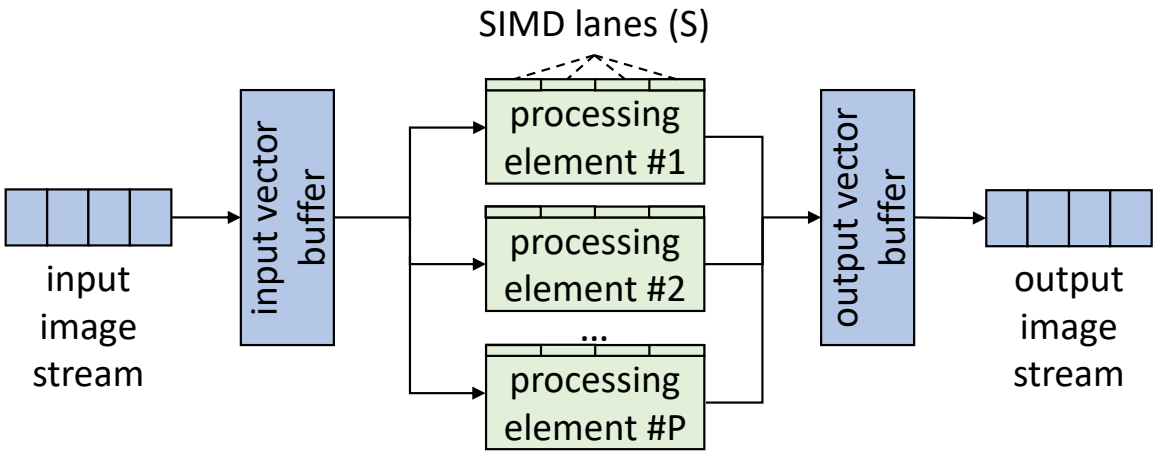


Figure 2.3.: Overview of the MVTU as in [31]

neural network, which contain the weights as well as perform the multiplication operation, while the SIMD lanes are the respective synapses of said neurons When using folding, the chosen number of PEs needs to be a divisor of the total number of neurons. The same applies to the number of SIMDs.

Various degrees of parallelism are implemented in other frameworks than FINN as well [36], [37], [38]. Venieris et al. [36] introduce FPGA reconfiguration for considering resource restrictions to mapping convolutional neural networks on FPGAs. Another implementation by Zhao et al. [38] sets the parallelism by controlling the number of input words per cycle and the resulting output streams. A completely different strategy that could be considered as time multiplexing is to implement an actual soft core architecture tailored to neural networks, such as VTA [39].

2.2. Side-Channel Attacks

Side-channel attacks in general exploit the correlation between calculations performed on the hardware and their a measurable physical property. During the calculations, transistors switch inside the circuit, resulting in a data-dependent switching activity. The attacker may, for example, measure the power consumption during the processing of sensitive data and uses it to extract the data. [25].

For side-channel attacks, different physical properties can be observed:

- Power analysis attacks: Measure the electrical power consumption of the device
- Timing attacks: Exploit variations in execution time
- Electromagnetic (EM) analysis attacks: Observe emitted electromagnetic radiation

A distinction is made between local and remote attacks. In local attacks, the adversary has physical access to the device and can directly perform measurements. In contrast, remote power attacks exploit shared hardware resources. For example, malicious logic can

be deployed on the same device to sense voltage fluctuations or abuse existing on-chip sensors. This is particularly relevant in multi-tenant environments, such as cloud FPGAs, where multiple users share the same hardware but operate in separate regions of the device. An attacker can place monitoring logic in their assigned region and observe activity from a neighboring accelerator. Common circuits used for sensing voltage fluctuations include Time-to-Digital Converters (TDCs) and Ring Oscillators (ROs).

Passive side-channel attacks infer secret information by analyzing variations in physical signals. They typically require knowledge about the implementation, such as the algorithm or the time window in which it executes. Three main analysis approaches are commonly used:

- Simple analysis: Manual analysis of collected traces
- Statistical analysis: Uses multiple observations and statistical methods to infer secret values
- Profiling attacks: Train a model on labeled traces, e.g., template or machine-learning attacks

Simple Power Analysis (SPA) directly interprets individual traces to identify data-dependent patterns. Statistical approaches usually rely on multiple measurements. For example, Differential Power Analysis (DPA) analyzes differences in power consumption caused by data manipulations and uses a selection function to identify correlated regions in the traces.

2.2.1. On-Chip Measurements

This thesis uses passive side channel measurements with remote measurement capabilities, as has been shown for FPGAs before [27]. Such attacks measure the voltage fluctuations while the victim circuit is operating on sensitive data [25]. These voltage fluctuations are then used to recover secret information from inside the circuit, typically using some supporting public information, such as ciphertexts in side channel attacks on cryptographic accelerators [25].

Due to fluctuations in the on-chip power distribution network, even digital programmable FPGA logic can be used to sense voltage fluctuations by regular users. The working principle of such sensors is to use a long path in which timing violations occur. Across that path, registers are connected. By that we have multiple registers as path endpoints with a gradually increasing delay. Because the delay of a transistor is inversely proportional to supply voltage, the extent of timing violations that occur in the registers is dependent on the local supply voltage. Thus, the more registers have violations, the lower the voltage, and vice versa.

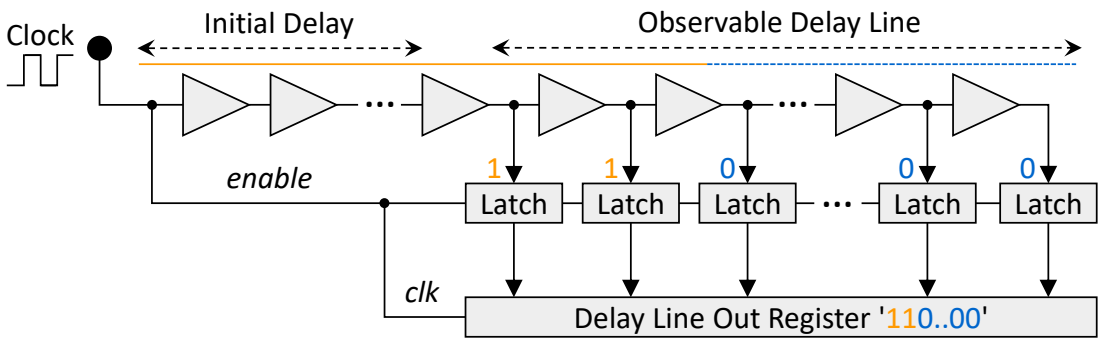


Figure 2.4.: TDC sensors as in [44].

2.2.2. Time-to-Digital Converters

In literature, it has been shown that carry-chains available in most FPGAs, can be used efficiently to implement such a voltage sensor, because of their low difference in delay between wires along the path to which registers can be connected [26]. To explicitly use them, in Xilinx FPGAs we can instantiate `CARRY4` or `CARRY8` elements. We can then floorplan them using `RLOC` and `BEL` constraints, generated dynamically in the used hardware description language. Since these sensors actually measure delay, they are also called Time-to-Digital Converters (TDCs). Such TDCs can then deliver measurements from which power analysis side-channel attacks [25] can be performed from inside the respective FPGA chip [27], [40].

Alternatively to the used sampling method, ring oscillators can also be used for measurements. However, they are typically slower in sampling speed [41], [42], and attacks based on them are easier detected and prevented in cloud environments [40], [43].

2.2.3. Routing Delay Sensors

Routing Delay Sensors (RDS) [45] are on-chip voltage fluctuation sensors that exploit the delay variability of FPGA routing resources to capture power side-channel information. Instead of relying on dedicated carry chains as in time-to-digital converters (TDCs), RDS leverage the propagation delay of the FPGA interconnect network, i.e., wires and routing multiplexers, which are also affected by voltage fluctuations in the power delivery network. As a result, switching activity of a neighboring circuit induces measurable delay variations that can be observed through carefully designed routing paths.

Earlier delay-based sensors such as TDCs use tapped delay lines implemented with carry chains and strict placement constraints to ensure uniform routing delays and high sensitivity. RDS follow a different design principle: rather than forming a strict delay line, multiple routing paths are created between a common input and a set of sampling flip-flops. Each path has a slightly different propagation delay, and voltage fluctuations modify these delays simultaneously. When sampled by a register bank, the captured bit pattern reflects the propagation depth of the clock edge through the routing network.

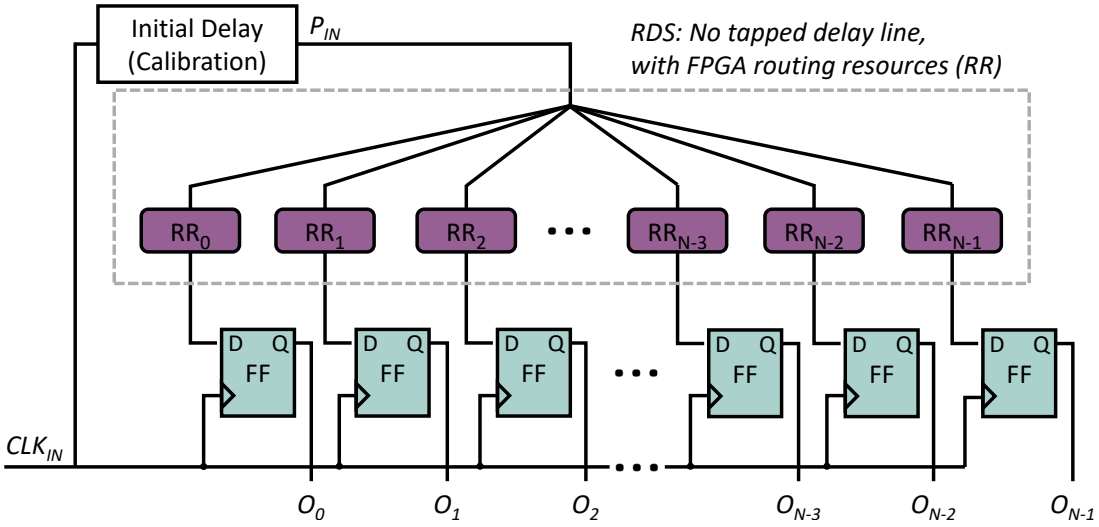


Figure 2.5.: Routing delay sensors as in [45]

Figure 2.5 illustrates the architecture of an RDS. A clock signal enters the sensor at node P_{IN} and propagates through multiple routing resources toward the inputs of N flip-flops. Because the routing delays differ slightly, the clock edge reaches each flip-flop at a different time. Sampling the register therefore produces a thermometer-like bit vector whose Hamming weight reflects the delay distribution of the routing paths. Voltage drops caused by the victim circuit change the propagation delays of these paths, leading to variations in the register output that can be recorded as side-channel traces.

In contrast to TDC sensors, RDS do not require a tapped delay line or strict placement constraints. Instead, the FPGA place-and-route tool is allowed to freely place the flip-flops and route the paths between the clock input and each register. This approach creates a large number of routing-dominated paths with slightly different delays, which increases the number of bits that toggle during voltage fluctuations and thereby improves the captured side-channel signal. Moreover, the absence of dedicated carry-chain structures makes RDS easier to deploy and harder to detect in multi-tenant FPGA environments.

2.2.4. Sensor Calibration

Proper calibration is required to ensure that the clock transition falls within the observable delay window of the sensor during sampling. The goal of calibration is therefore to adjust the phase shift between the propagated clock edge and the sampling clock such that the register output exhibits high sensitivity to delay variations.

Calibration is implemented by introducing a configurable initial delay before the clock signal enters the routing network. This delay line typically consists of coarse delay elements (e.g., LUT-based delay stages) and fine delay elements (e.g., carry-chain segments).

For sensors with tapped delay lines, the calibration objective is to position the clock edge approximately in the middle of the observable delay line, resulting in a register value with

a Hamming weight close to $N/2$. In this configuration, both increases and decreases in propagation delay can be observed as changes in the register output.

For RDS, the calibration procedure instead maximizes the variance of the sensor output. The initial delay is incrementally increased until all register bits evaluate to logical “1”, indicating that the clock edge lies outside the observable delay window. The delay is then gradually reduced using fine delay adjustments until the output begins to contain both zeros and ones. At this point, a subset of flip-flops samples the clock edge close to their switching threshold, making them highly sensitive to delay variations caused by voltage fluctuations.

2.3. Side-Channel Attacks on AI Hardware

In recent years, neural network accelerators have increasingly become targets of physical side-channel and fault attacks [28], [46]. This trend is driven by the widespread deployment of FPGAs for machine learning acceleration in cloud environments [47], where remote physical attacks are feasible [27], [48]. These attacks either target privacy-sensitive input or output data [29], [49], [50], or aim to reverse engineer neural network intellectual property, including network topology, weights, and other model parameters [28], [30], [46], [51], [52], [53], [54].

Despite different goals, these attacks share many technical similarities across passive side-channel and active fault injection techniques. Fault attacks can be used to corrupt predictions [50], but also enable reverse engineering of model parameters [53], [54]. Both passive and active attacks [27], [48] can be implemented remotely using standard FPGA primitives, extending the threat model beyond scenarios with direct physical access.

2.3.1. Model Extraction Attacks

The reverse engineering of neural networks aims to recover both architecture and parameters, including the number of layers, neurons, activation functions, weights, and biases. For convolutional networks, additional layer-specific parameters must also be inferred. Such attacks threaten the intellectual property of model owners and can even reveal private inputs such as images.

A full model extraction attack is typically performed iteratively by first recovering the number of neurons in a layer, followed by weights, layer boundaries, layer assignments, activation functions, and output values, which are then used as inputs to attack subsequent layers. In general, attacks either aim to reconstruct the exact target model or to obtain a substitute model with similar accuracy. For example, Yu et al. [52] combine electromagnetic analysis with parameter guessing and adversarial refinement to approximate the target model.

Activation functions can be identified from their side-channel signatures. Prior work shows that functions such as ReLU, sigmoid, tanh, and softmax exhibit distinct timing, power, or electromagnetic patterns, enabling their identification through profiling, waveform analysis, or machine learning [30], [55], [56], [57].

The number of neurons and layers can often be inferred from repeating patterns in side-channel traces. Sequential neuron executions and characteristic voltage or power variations reveal layer boundaries and neuron counts, either through direct analysis or machine learning-based classification [30], [51], [55], [58].

Recovering weights and biases is more challenging but has been demonstrated in multiple settings. Correlation-based attacks, timing analysis, and quantization-aware methods enable extraction from software and FPGA implementations, often proceeding layer by layer using previously recovered outputs [55], [56], [59]. In addition, fault attacks can reveal weight information by analyzing deviations between faulty and correct computations [53], [60].

Overall, while architecture parameters can often be recovered reliably, weight extraction remains significantly more difficult, particularly for highly parallel FPGA-based accelerators.

2.3.2. Input Reconstruction Attacks

Side-channel leakage can also reveal private input data. Wei et al. [49] show that power consumption during convolution correlates with processed pixel values, enabling image recovery without knowledge of the network architecture. They reconstruct images either by separating foreground and background pixels using power thresholds or by estimating pixel values through power templates.

Moini et al. [61] demonstrate remote input recovery on binarized CNN accelerators using TDC-based power sensing. After averaging and denoising the traces, they reconstruct recognizable images and evaluate the results using normalized cross-correlation and structural similarity. Other approaches transform power traces into 3D power surfaces, which are then processed by a neural network to recover the original input [62].

Dong et al. [63] further exploit timing information derived from power traces for image reconstruction, assuming knowledge of the input size and the number of neurons in the first hidden layer.

Generative models can also enhance side-channel attacks on less conventional systems. For example, conditional generative adversarial networks have been used to recover magnetic resonance images from noisy power measurements of analog compute-in-memory systems, demonstrating robustness against noise-based countermeasures [64].

2.4. Functional Safety

2.4.1. Out-of-Distribution Detection

A critical limitation of Neural Networks (NNs) is their inability to reliably identify inputs that lie outside the training distribution, commonly referred to as out-of-distribution (OOD) inputs [65]. When encountering such inputs, NNs often produce overconfident yet incorrect predictions, which is particularly problematic in safety-critical applications. Therefore, detecting and properly handling OOD inputs is essential for deploying NN accelerators in real-world systems. This directly supports the *fail-safe* property, where a system either produces trustworthy predictions or flags uncertain outputs. In the following, we briefly review representative methods for OOD detection that do not require retraining or modifying the original model.

A simple baseline is the Maximum Softmax Probability (MSP), which detects OOD samples by thresholding the highest softmax output:

$$f(x) = \max_i \left\{ \frac{e^{x_i}}{\sum_j e^{x_j}} \right\} \quad \text{for} \quad i = 1, \ldots, n. \tag{2.1}$$

However, NNs are known to produce overconfident predictions even for OOD samples [66].

An alternative is MaxLogits, which uses the maximum pre-softmax output:

$$f(x) = \max_i \{x_i\} \quad \text{for} \quad i = 1, \ldots, n. \tag{2.2}$$

The intuition is that in-distribution (ID) samples yield higher logits, while OOD samples result in lower confidence.

Entropy-based methods compute the uncertainty of the softmax distribution:

$$f(x) = -\sum_{i=1}^{n} x_i \log(x_i). \tag{2.3}$$

Low entropy indicates confident predictions for ID data, whereas high entropy suggests OOD inputs [67].

Energy-based approaches instead model the data distribution using an energy score:

$$f(x) = -t \log \left(\sum_{i=1}^{n} e^{\frac{x_i}{t}} \right). \tag{2.4}$$

OOD samples typically exhibit higher energy, although performance may degrade for subtle distribution shifts [68].

ODIN [69] improves separability by combining temperature scaling with small input perturbations, increasing confidence for ID samples while reducing it for OOD inputs.

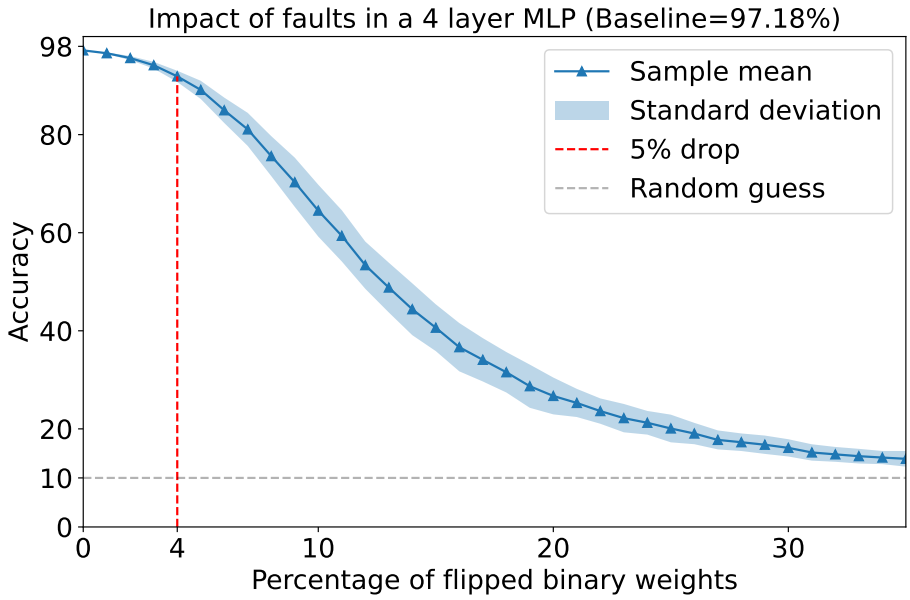


Figure 2.6.: Monte Carlo simulation (100 samples) of weight bit flips in a 4-layer neural network.

Despite their effectiveness, these methods share key limitations. They are often not designed for efficient hardware implementation and typically operate after inference, introducing additional latency. This motivates the development of lightweight and concurrent OOD detection mechanisms tailored to hardware accelerators.

2.4.2. Fault Detection in AI Hardware

Effective detection of runtime faults in NN accelerators is essential to ensure reliable and correct operation. The impact of such faults on model predictions is illustrated in Figure 2.6, which shows the degradation in accuracy caused by random weight bit flips. Traditional fault detection techniques, such as Error-Correcting Codes (ECC) [70] and hardware redundancy schemes including Triple-Modular Redundancy [71] or Cyclic Redundancy Code (CRC) [72], exhibit notable limitations. While ECC can effectively correct single-bit errors in memory, it does not protect against faults occurring within neural processing logic. Redundancy-based approaches provide stronger protection but incur significant hardware overhead, making them impractical for resource-constrained accelerator designs.

To address these challenges, several fault detection mechanisms tailored to NNs have been proposed. In [73], a secondary redundant network is used to detect discrepancies in the primary model's output by comparing predictions during runtime. Although effective, this approach introduces overhead due to the need to maintain and execute an additional model. Activation Range Supervision [74] detects bit-flip faults in CNNs by monitoring neuron activation values and identifying deviations from predefined ranges. However, this method requires architectural modifications, limiting its applicability to existing deployments.

Other approaches focus on detecting adversarial weight faults using signature-based techniques. RADAR [75] employs checksum-based 2-bit signatures over interleaved and masked weight groups for real-time fault detection and recovery. Similarly, Javaheripi et al. [76] propose a hash-based method that compares precomputed and runtime signatures of sensitive layers to detect integrity violations. These methods, however, are limited to detecting adversarial faults in weights and require additional hardware for signature generation and verification.

In contrast, this thesis approach targets a gray-box setting without requiring model duplication or detailed knowledge of the hardware implementation. It enables real-time detection of faults in both model parameter memory and neural processing logic, and is specifically designed for binary neural networks while introducing only minimal hardware overhead.

Part II.

Contributions

3. Reverse-Engineering Attacks on AI Hardware

3.1. Reverse Engineering Neural Network Folding with Remote FPGA Power Analysis

The work described in this chapter was published in [3] and is joint work with co-authors Dennis Gnad and Mehdi Tahoori.

This work investigates passive power side-channel attacks on folded neural networks implemented on Field Programmable Gate Arrays (FPGAs). It shows that existing profiled attacks are insufficient to fully recover the number of neurons or the layer topology of folded architectures, even when trained on data from the same implementation.

To address this, a two-stage attack is proposed. First, a time-series based approach is used to recover the layer topology, including folding parameters. In a second stage, this information is leveraged to accurately reconstruct the number of neurons in each layer, demonstrating that knowledge of folding is essential for precise reverse engineering.

In contrast to prior work, this approach follows both machine learning and profiled side-channel analysis best practices, including cross-validation and the use of physically separate devices for profiling and evaluation [77], [78]. Additionally, the robustness of the attack is evaluated under varying environmental operating conditions.

The main contributions are as follows:

- Attack on a practical neural network framework fully mapped to an FPGA
- First recovery of folding parameters in time-multiplexed neural network implementations with overlapping layer execution
- Reverse engineering of neuron counts across all layers based on recovered folding parameters
- First machine learning-based profiled side-channel attack considering separate profiling and target devices
- Demonstration of attack robustness under varying environmental conditions

The remainder of this section is taken directly from [3] and is reproduced verbatim.

3.1.1. Methodology

This work follows the threat model that has already been proposed in previous work on passive voltage-based side-channel attacks in multi-tenant FPGAs [27], [40]. Here, we focus on attacking neural network accelerators [29], [30], [49], [54], instead of cryptographic modules. The assumption is an on-chip attacker who has control over part of a multi-tenant FPGA, i.e. a single FPGA chip in which multiple users have their own dedicated partial reconfiguration regions. In each of the separated regions, a single user (attacker) can run their own design, but without being able to directly connect to another user (victim) of the FPGA. To perform an attack, the TDC sensors of the attacker and the neural network accelerator of the victim are logically isolated and the attacker has no control over the random inputs which are sent to the FPGA separately. The attacker then aims to recover secrets of the neural network, solely based on the measurements from the TDC.

It is assumed that the attacker knows the start of the inference as the TDC traces show a visible voltage drop on the start of the inference. For attacking individual layers it is required to split each trace into traces for each layer. This requires recovery of the layer boundaries or knowledge about the period of time during which the layer is active, which has been achieved by Zhang et al. [30] before for unfolded networks. We implemented the method used in their paper and found that it also successfully recovers the layer boundaries of folded layers in FINN. Sometimes, minor offsets occur in layer recovery, however our following results are usually not influenced by that. Nevertheless, to be able to present absolutely consistent results, we stick to the initially recovered layer boundaries. While layer boundaries do not really exist in FINN, as all layers are active at all times, we can still log the first and last time a layer receives input from a previous layer and use that for layer segmentation.

We will show that all different neuron foldings for the examined neural networks can be successfully recovered. As we set the same folding-factor for neuron and synapse fold for the generated networks, the synapse fold could potentially also be attacked. However, for our follow-up attack on the number of neurons in a neural network, the synapse fold is of less interest, as it does not affect the number of active neurons during one cycle and therefore it will not be covered here.

In detail, we then have the following assumptions:

- The Inputs to the network are unknown and random
- The start and end time of each individual inference is known
- The networks deployed on the *profiling* device are partially known:
  - The number of layers, neurons and folding can be controlled by the attacker
  - The layer boundaries can be recovered and are known during training
- For the network running on the *target* (i.e. victim) device:
  - The number of layers were previously seen in the profiling device

Table 3.1.: Combinations of neural networks and folding types implemented and evaluated in this work

Network	Folding Parameter								
	2x	4x	5x	8x	10x	16x	20x	32x	64x
MLP-16	✓	✓		✓		✓			
MLP-20	✓	✓	✓		✓		✓		
MLP-32	✓	✓		✓		✓		✓	
MLP-64	✓	✓		✓		✓		✓	✓
CNN	✓	✓		✓		✓		✓	✓

– As accelerators generated by FINN are deterministic, we assume the same layer boundaries for inferences on the target device.

In summary, we perform a profiled side-channel attack without known inputs or outputs, as the adversary can only measure the voltage fluctuations during the inference, but not influence it, which was also performed by Zhang et al. [30]. However, they did not test their attack on another device, which is typically required in profiled side channel attacks [77], [78]. Such attacks use a profiling device on which knowledge is available to create a profile, in the scope of this and related work those are the model parameters of the neural network that shall be reverse engineered. Opposed to other works, our actual attack uses the created profile on a different physical device (target), which will be used to measure the *test* set in machine learning terms.

In Table 3.1 the 26 setups used in the attack are shown. As not all folding types can be applied to all networks, checkmarks (✓) denote combinations of network and folding, which are included in the experiments.

3.1.1.1. Profiling and target devices for training and testing

Physical properties of devices are never exactly the same due to the manufacturing process causing slight variations between all produced ICs, even in the same batch. Thus, to prove applicability, profiling side channel attacks are tested with at least two distinct devices, of the same type. One device is set up as the profiling device on which we collect data to train our model, i.e. classifiers. Subsequently, we use the other device as the role of the victim device to which the adversary has only limited access.

We perform training and cross validation on the profiling device, and separately test the classifier with freshly collected data from the testing device. Like that, we follow established evaluation practices for both machine learning and profiled side-channel attacks, which is not considered in previously published profiled attacks on neural networks.

Table 3.2.: Characteristics of the top 10 features utilized in classifiers for folding detection in MLPs and CNN with descriptions as explained in [79].

Feature	Description	Variants
FFT Coefficient	Fourier coefficients of the one-dimensional discrete Fourier Transform	angle: 23, 52, 85; abs: 22, 56; imag: 56
AR Coefficient	Unconditional maximum likelihood of an autoregressive AR(k) process	1_k_10
Autocorrelation	Autocorrelation of the specified lag, according to the formula	lag 1
Quantile	q quantile of x	q: 0.1, 0.2
Change Quantiles	Average, absolute value of consecutive changes of the series x with a fixed corridor given by the quantiles ql and qh of the distribution of x	var, isabs: False, qh: 0.4, ql: 0.2; var, isabs: True, qh: 0.4, ql: 0.2; mean, isabs: True, qh: 0.4, ql:0.0; mean, isabs: True, qh: 0.4, ql:0.2

3.1.1.2. Feature selection

In most of the experiments, we use the popular python package *tsfresh* for identifying and extracting meaningful features from time series data [79]. The most comprehensive previously published attack on reverse engineering neural network model parameters also uses features based on tsfresh successfully [30].

The tsfresh library computes a total of 794 features by utilizing 63 time series characterization methods. Considering classification tasks, instead of mapping all data points from a time series to a class, specific time-series features can be extracted, such as distributions, entropy, correlation properties, Fourier transformed features, just to name a few. This approach can improve generalization of trained models by removing irrelevant properties from the training set and these features are typically also designed to be more invariant of specific time offsets. The extracted features can then be used in combination with any machine learning classifiers.

For the conducted results on folding, we use the top 10 features, while our results on neuron recovery use up to 50 features. A list of the features which will be the input for the folding detection classifier for MLPs and the combined classifier for MLPs and the CNN is given in Table 3.2. The features listed in this table are calculated for different sensors, resulting in 10 features for each network.

3.1.1.3. Choice of classifiers

In this paper we chose the supervised classification technique of Random Forest Trees [80] for the classification of folding and number of neurons. This technique is an *ensemble learning method*, which decides its output by majority vote of multiple decision trees, and has been successfully used previously [30].

In our experiments, we also tested other machine learning methods like Neural Networks, i.e. Multi-layer Perceptrons (MLP) [81], as well as AdaBoost [82], GradientBoosting [83], but found that Random Forest Trees performed better than the others for our classification tasks.

Furthermore, we train classifiers with varying numbers of sensors and compare them with each other. For this, a classifier is trained on each sensor separately and one classifier is trained on all sensors. Then the cross-validated accuracy score of the classifiers is compared.

3.1.2. Experimental Setup

Our experimental setup is based on two samples of the Xilinx Pynq-Z1 board, one as profiling device and the other as target/victim device respectively, to follow our methodology. The Pynq-Z1 is one of the main devices supported by the FINN compiler. It features a Z-7020 FPGA SoC with two ARM cores and Xilinx Artix-7 FPGA fabric, containing about 85k logic elements. Currently, the largest server-grade PCIe FPGA accelerators are available with about 6× as much logic elements. Therefore, even on those FPGA devices *folding* is required to fit larger networks. Inside this platform, we implement both our attacker and victim design, considering an on-chip attacker which can integrate TDCs, but has no other direct (digital) connections to the victim. The ARM processors are mainly used to control our experiments and store power traces in flash memory for offline analysis later.

FINN offers an exemplary MLP for the MNIST classification task, which we use in our experiments. Additionally, we implement a CNN for the CIFAR-3 dataset, which is similar to an exemplary network for the CIFAR-10 dataset. The MLP network consists of a thresholding layer followed by 3 hidden layers with 64 neurons each, and an output layer with 10 neurons for the output layer, as was presented in [31]. The thresholding layer is an activation using the identity function with the required bit width to quantize inputs. In this example provided by Xilinx all layers use the identity function for activation. Quantization reduces the precision to 2-bit activation and weight, resulting in ternary weights: $\{-1, 0, 1\}$. We trained three additional MLPs with the same architecture, but with 32, 20 and 16 neurons in each hidden layer, and quantized it with the FINN component *Brevitas*. Our trained CNN network is shown in Figure 3.1.

In Figure 3.2 an overview of the implementation is given. The embedded ARM core sends images through AXI bus DMA to the implemented FINN Accelerator, which sends the prediction results through another DMA on the AXI bus. Helper signals are utilized which

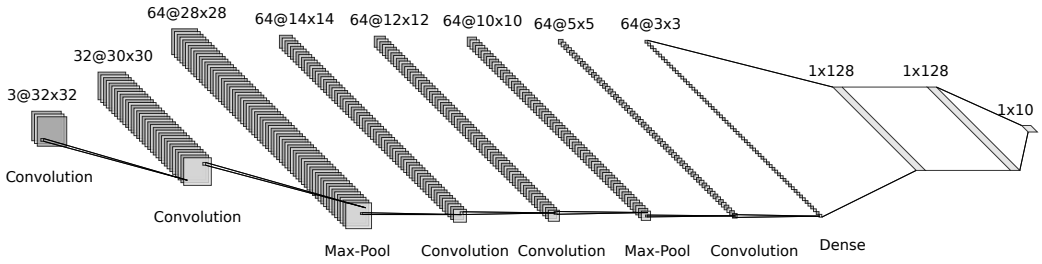


Figure 3.1.: Convolutional network architecture.

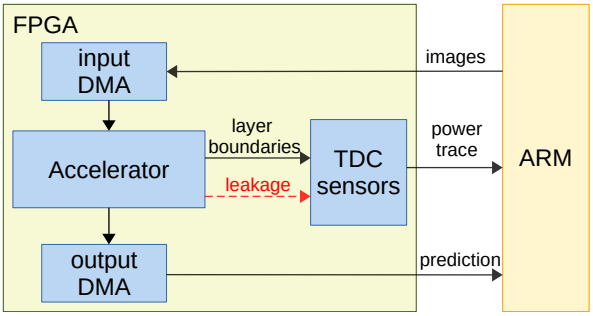
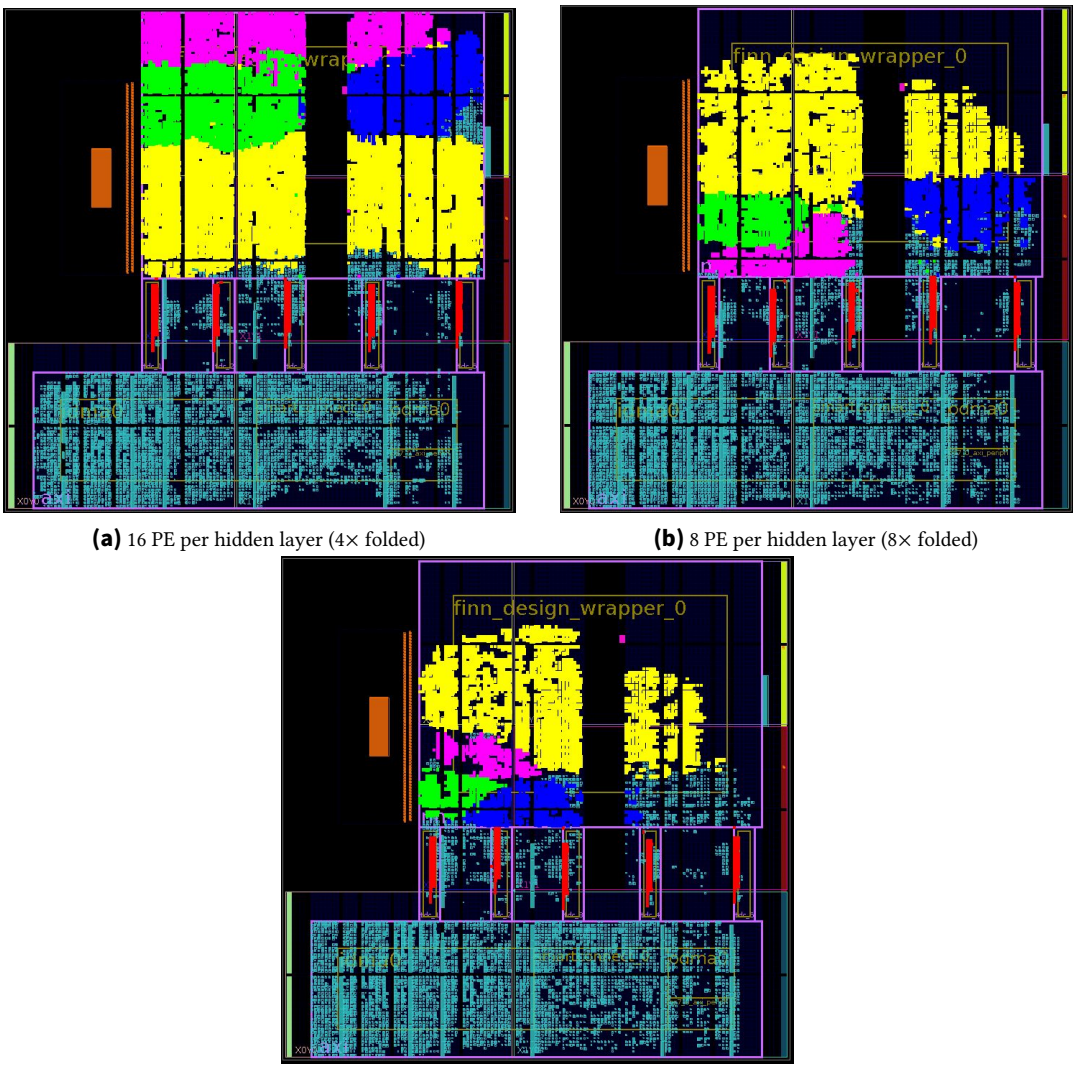


Figure 3.2.: Setup consisting of the accelerator, TDC circuit, helper signals for layer boundaries and the ARM processor

reflect the recovery of layer boundaries by logging the point of time of handshake signals for data transactions between layers. The logic in the TDC sensors uses these signals to start and stop recording voltage fluctuations into on-chip BRAM. These power traces are then later transmitted to the ARM core, again via AXI bus.

The respective floorplans of three accelerator setups with TDCs, accelerator and AXI module can be seen in Figure 3.3, all showing the same neural network with different folding parameters. For measuring layer activity, TDCs are spread out horizontally below the accelerator. Remaining control logic, and AXI bus are below the TDCs and the neural network is above the TDCs. The total FPGA design in Figure 3.3(a) utilizes about 74.50% of slices, which can be reduced to 59.78% with the floorplan shown in Figure 3.3(b). It further reduces to 52.50% as is also indicated in the floorplan in Figure 3.3(c). For our initial measurements, 100× averaging is applied to the power traces by running 100 inferences of different images and then taking the average of the sum of all traces. We smooth the traces with an additional Kalman filter for visualization in Figure 3.4, which will be detailed in our results section. The TDCs run at 200MHz and the accelerators at 100MHz. 1000 samples are collected for each hidden layer, which results in 3000 samples per network configuration for the MLPs and a reduced number of 200 for the CNN. A total number of 63600 samples are collected for which 50880 (80%) are used for training and 12720 (20%) are used for testing. After collecting the power traces and combining them into a data frame with different time series, one for each sensor, the selected top features are extracted from training data. The same top 10 features are then applied for feature extraction from data collected on the target device.



(a) 16 PE per hidden layer (4× folded)

(b) 8 PE per hidden layer (8× folded)

(c) 4 PE per hidden layer (16× folded)

Figure 3.3.: Exemplary floorplans of folded neural networks mapped onto the FPGA. **Red:** TDC sensors; **Cyan:** Control logic, BRAM, AXI communication; **Yellow:** 1st layer; **Violet:** 2nd layer; **Green:** 3rd layer; **Blue:** output layer.

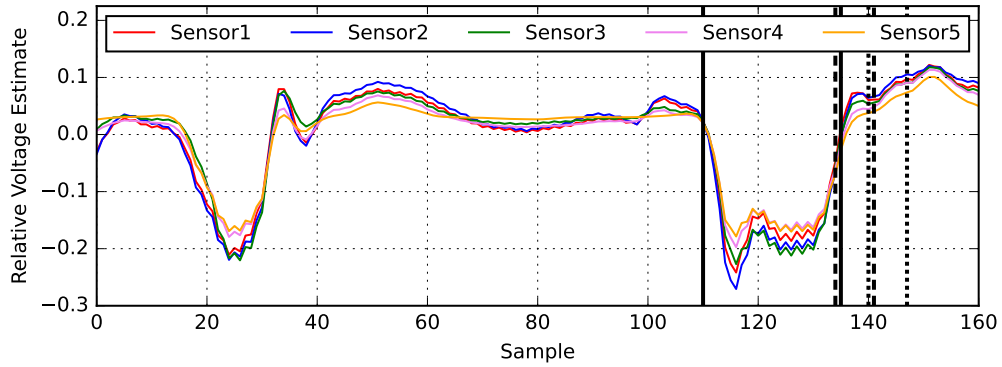
3.1.3. Results

As a preliminary result, we present how folding affects the amount of power consumption and voltage fluctuations on the FPGA. In Figure 3.4 we show the respective power traces of an unfolded, and two types of folded networks. The y-scale denotes the normalized value of each TDCs output after min-max feature scaling with a range of [-1,1].

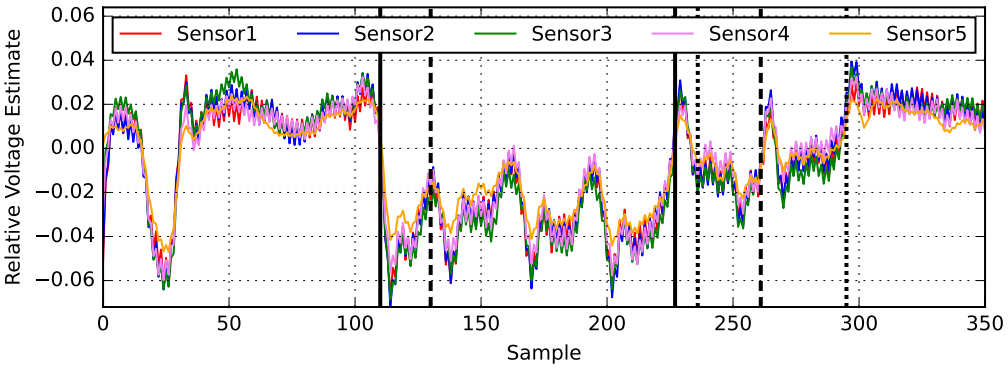
Without neuron folding, we show an MLP (i.e. fully-connected neural network) with 3 layers and 16 neurons per hidden layer, running inference, averaged over 100 runs in Figure 3.4(a). However, synapse folding (16x), had to be applied on the first layer of this network, because of limited resources on the device. The first layers computation can be

seen in the trace between time sample 110 and 135, which resembles a pattern that we can also observe repeatedly in the trace of the folded layers. For the following layers this is not as easy to distinguish just by looking at the traces.

For power traces of the two folded MLPs, we exemplary show a 64-neuron MLP with a respective 4× folding in Figure 3.4(b) and 8× folding in Figure 3.4(c). When folding is applied, FINN uses pipelining to reclaim some of the performance lost from sequential execution [31]. Due to this design, all layers perform some neuron computations that overlap in time with the computations of their subsequent or preceding layer. We mark the respective starting and ending times of all the layers with vertical lines in Figure 3.4. The start times at which each layer first receives data and the end times in which each layer last sends data are marked in a different style for each layer. We assume such pipelined operation can mislead simple power analysis attacks. Furthermore, we can also observe that the traces highly differ in magnitude and length, which a simple pattern detection could not clearly differentiate. That suggests using a machine learning-based classifier to automatically distinguish different folding parameters, before more details can be recovered about the network.

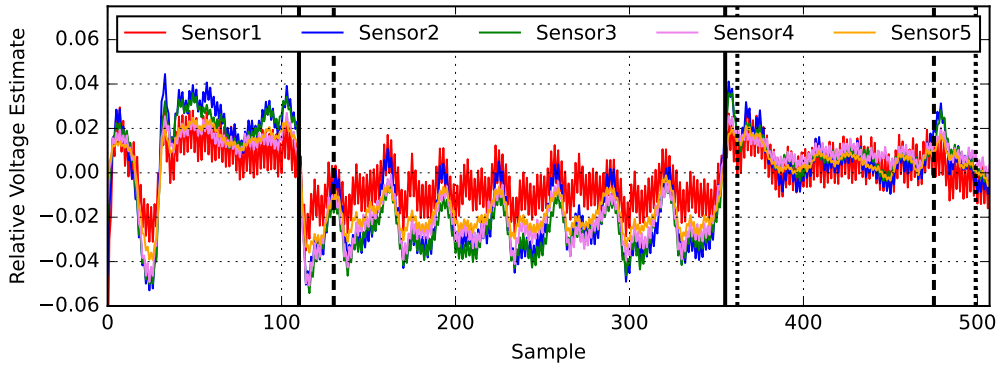


(a) 16 PE unfolded network, with overlapping layer activity

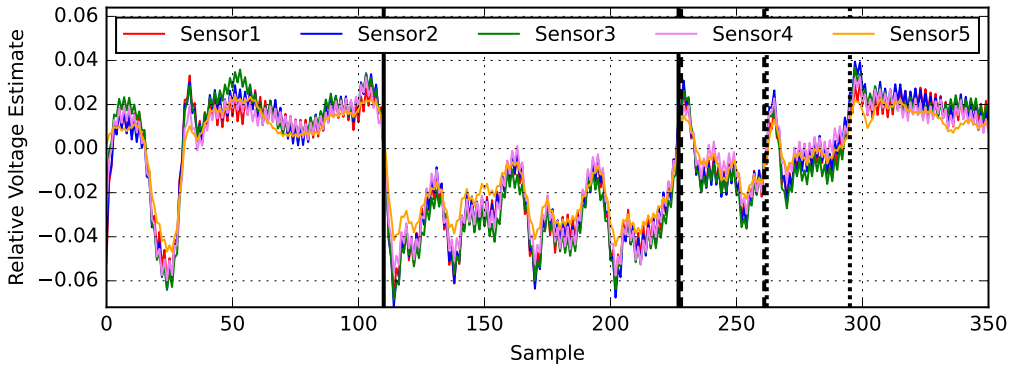


(b) 16 PE per hidden layer due to 4× folding applied, with overlapping layer activity

Figure 3.4.: Comparison of power traces collected from networks with different neuron folding. Each hidden layers activity is taking place between the vertical lines of the same style. 1st layer: —; 2nd layer: - - -; 3rd layer: ▪▪▪▪▪



(c) 8 PE per hidden layer due to 8× folding applied, with overlapping layer activity



(d) 6 PE per hidden layer due to 4× folding applied, with our segmentation for classifier training

Figure 3.4.: Comparison of power traces (continued).

While all layers are always active at all times in a FINN-generated network we can still determine layer boundaries and split the entire power trace into a chunk of trace for each respective layer. The dotted lines in Figure 3.4(d) mark the last time that data is received by the corresponding layer and the last time the following layer receives data. Thus, we know that between these cycles the first layers neurons are definitely active. The same procedure is repeated for identifying the following layers activities and the power trace is split accordingly. These individual traces, as the ones marked in Figure 3.4(d), are then used to train classifiers and perform the folding detection.

Note that while we split the trace by layers, the computations of different layers are still present in each power trace due to the mentioned overlap from pipelining. This sets our method apart from previous works by Zhang et al. [30] or Tian et al. [51], which only consider sequential computations or multiple inferences at once, but not parallel (pipelined) layer activity from the same network.

3.1.3.1. Initial Neuron Recovery Results

To show that reverse engineering folding parameters is a necessity, we train a Random Forest classifier for an unfolded network similar to the work done in [30] for networks

with #neurons 16, 20 and 32. The classifier achieves a cross-validation accuracy of **100%** on unfolded layers. However, when trying to predict the number of neurons for folded layers, the classifier only achieves an accuracy of **48%** on the profiling device and **44%** on the target device. We conclude that a classifier trained on unfolded layers, as in the work by Zhang et al. [30], cannot efficiently work on the networks generated by FINN, if folding is applied. The classifier is then trained on data from layers with various foldings and #neurons. Similar to [30] we compare the results for different sensors and a different number of sensors, for which we could see distinct behavior as Figure 3.4 indicates. When utilizing only a single sensor, the highest accuracy that could be reached for this classifier was **82%**. By combining the sensors time series into a single data frame, the cross-validation accuracy can be increased to **85%** with the top 10 features from tsfresh, which is not further improved with more features.

3.1.3.2. Folding Recovery on Multi-Layer Perceptrons (MLPs)

For fully-connected layers in MLPs, training data is collected only from running the 64-neuron and 20-neuron network on the profiling device. Figure 3.5 shows the accuracies with 5-fold cross validation for the Random Forest Tree classifier trained on the features extracted with tsfresh and raw data. The classifier trained on the 10 chosen features achieves an cross-validation score of **100%**, while the classifier trained on raw data only achieves an average score of **88.97%**. Furthermore, we can also show that a classifier solely trained on MLPs is not sufficient for classifying folding on CNNs, with the accuracy dropping to 36%. The classifier trained on CNN and MLP overcomes this issue and achieves 100% cross-validated accuracy score for both cases.

For the next step, the classifier is tested with samples collected on the target device. In Figure 3.6 it can be seen that for both, the 64- and 20-neuron network, for which the classifier was originally learned, and the unseen 32-neuron network, an accuracy score of **99-100%** can be reached on the target device. We also include a small network with only

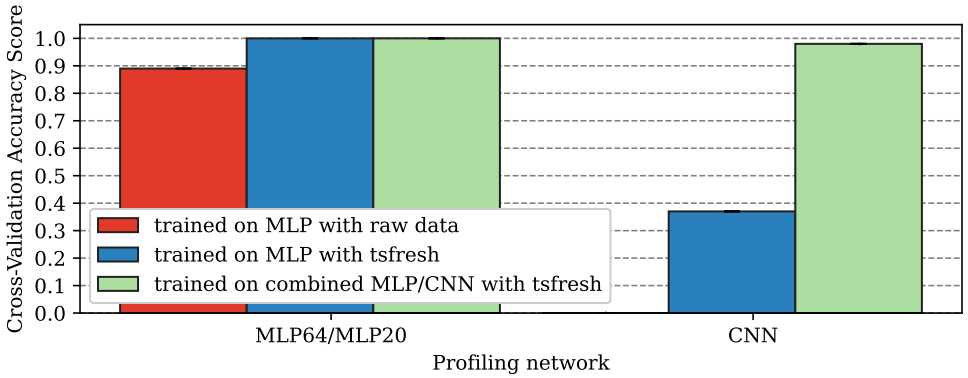


Figure 3.5.: Cross-validation accuracies for folding detection from the three trained classifiers. The red bar is for the classifier trained on raw data from the MLP, blue for the classifier trained on extracted features from the MLP data and green for the combined classifier for MLP and CNN.

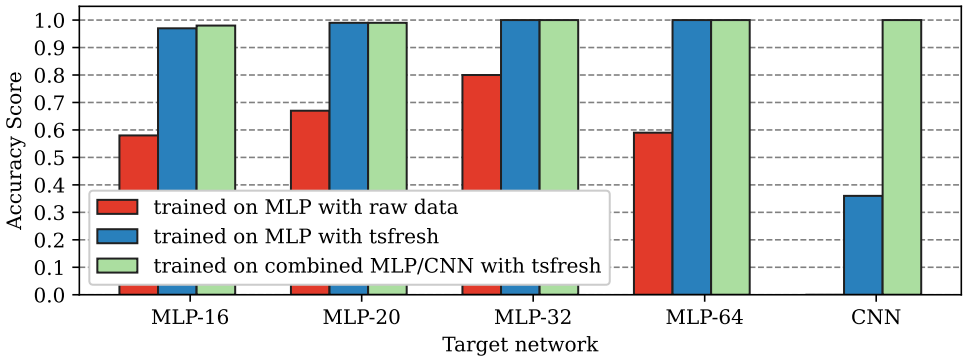


Figure 3.6.: Folding detection accuracies on the target networks from the three trained classifiers. The red bar is for the classifier trained on raw data from the MLP, blue for the classifier trained on extracted features from the MLP data and green for the combined classifier for MLP and CNN.

16 neurons per layer in the experiment and still achieve an accuracy of **98%**. This result shows that folding parameters can be learned independent of the device and number of neurons per layer, so a different network can be utilized for training to attack a target device.

3.1.3.3. Folding Recovery on MLP and CNNs together

In practice, convolutional neural networks are widely deployed in more challenging classification tasks, typically in image processing. Therefore, a practical attack also needs to consider such networks, and we thus also apply our approach to a CNN.

To include both types of networks, we train the Random Forest Classifier on the power traces from the MLP as well as data from the CNN, to get a combined classifier. The resulting classifier achieves an accuracy of **98%** as displayed in Figure 3.5. In a test on the target device with combined power traces from the MLP and CNN we could achieve **100%** accuracy on the CNN as displayed in Figure 3.6. This shows that it is not necessary to train separate classifiers to detect folding for both, fully-connected and convolutional NNs, but a single classifier can be sufficient for both.

3.1.3.4. Recovering number of neurons

Using the classifier from subsubsection 3.1.3.1 on data from the target device, a maximum accuracy of **79%** is achieved using 30 tsfresh features, detailed in Figure 3.7. In this setup we aim to improve on the 85/79% neuron recovery by using the folding recovery information.

For that, we train a classifier that distinguishes between 16, 20, 32 and 64 neurons in any layer of the network with their respective foldings as shown in Table 3.1. Figure 3.7 displays the accuracy with 5 fold cross validation for training with 10 features in the first two bars of the plot “10(CV)”.

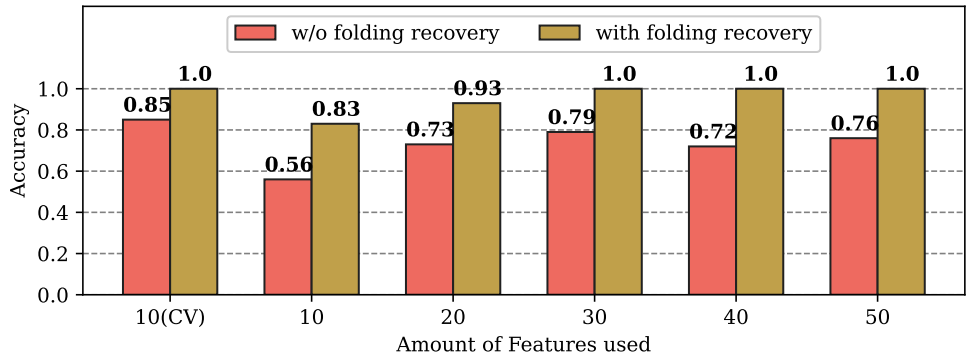


Figure 3.7.: Accuracy of recovering neurons on our target device for 16, 20, 32, 64 neuron MLPs, based on the folding detection of 4x folding from our combined classifier. For comparison the cross-validation accuracy of the classifiers trained with 10 features is given in the first column.

We showed before that it is possible to extract the neuron fold of a networks layer. This information can be used to reduce the combinations of #neurons and foldings that a classifier has to be trained on by identifying the folding before recovering the #neurons. Now specialized classifiers can be trained separately for each folding which is present in our experiments, reducing the number of classes to detect to multiples of the respective folding. Figure 3.7 also shows an example of the neuron recovery when a neuron fold of 4 could be identified for the layer. By recovering the neuron fold before the #neurons, we can drastically increase the accuracy even more, achieving 100% recovery.

3.1.3.5. Folding detection at different operating conditions

As a previous study by Heuser et al. [84] shows, changing environmental conditions, like temperature, can directly affect the power consumption of a device. Thus, the voltage fluctuations measured by the TDC are additionally dependent on temperature, which influences the efficiency of the classifier. In general, it is assumed that the profiled and attacked device are subject to the same environmental conditions, but this is not always the case in practice, especially for remote attacks in cloud environments. For a reasonable approach to experimentally validate the attack and recovery success under different environmental conditions, we also test our classification under different operating temperatures. We use the previous classifier, which is based on data measured at uncontrolled office room temperatures, around 18-25°C of the profiling device, and then apply it to the target device at various temperatures.

Results of the experiments for taking measurements on the target device at room temperature in a normal office space and for 0°C, 40°C and 70°C in a shielded climate chamber are presented in Figure 3.8. The results indicate that the accuracy for each target network decreases when the temperature gets higher compared to the temperature used during classifier training. The worst case result is 93% on the 20-neuron neural network compared to 100% at room temperature. In all cases, we still achieve over 90% overall and still 99%

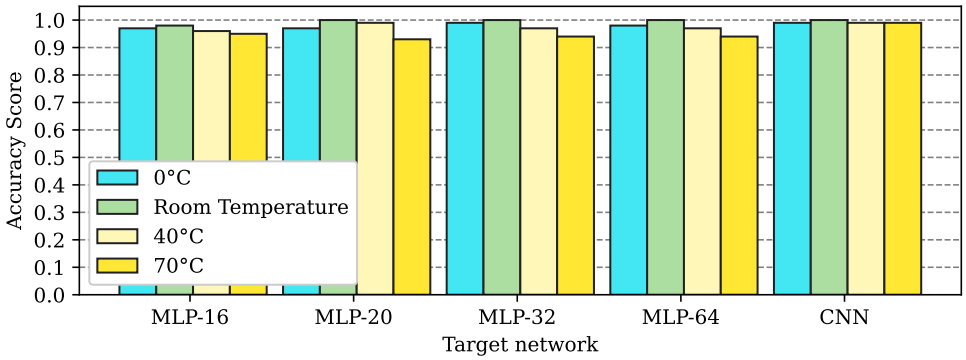


Figure 3.8.: Folding detection accuracy scores of the combined classifier for MLP and CNN under different temperatures of the target device. The classifier is trained on a profiling device at room temperature (18-25°C).

accuracy on the target CNN. The trained classifier still performs well, even when the neural networks and TDCs are running at the temperature limits of the used FPGA platform.

3.1.4. Section Discussion

Our results on recovering the number of neurons show that it is necessary to consider folding in side-channel attacks when more realistic neural network implementations on FPGAs are attacked. However, our results show that in the end folding is just a minor obstacle to attackers, and attacks are still possible when combining our results with previously published works in [30], [51], [54]. Therefore, it is still an open research question to find suitable countermeasures to prevent theft of neural network IP cores in multi-tenant environments or systems to which physical access for power analysis attacks exist. Even more, in realistic scenarios where resources are limited, such as folded implementations, countermeasures need to be light on resource usage.

For attacking privacy related data (i.e. inputs), a specific masking countermeasure has already been presented in [46]. For reverse engineering, a very expensive countermeasure that fully masks the entire network has been shown in [85], requiring almost 6× the resources of the protected network. Otherwise, only generic hiding side-channel countermeasures have been shown for Multi-Tenant FPGAs so far in [86], which still needs at least 2× the original resources and has not been evaluated on neural networks yet. Adding such countermeasures to resource constraint embedded devices will make them require even more folding, or make an implementation entirely impossible. Thus, research on more efficient countermeasures to prevent model stealing attacks is still needed.

In this work, we have shown that folding is an important aspect when considering profiled and machine learning-based side-channel attacks on neural network accelerators, which makes previous attacks harder to perform. Subsequently, we show how folding can be recovered before we perform recovery on the number of neurons. Like that, we can extend the neuron recovery from about **44%** to **100%**. Even when the target device is running under various elevated thermal conditions that were never included in the trained model,

our folding recovery still reaches accuracies between **93-99%**. By that, this work adds a missing piece to model stealing attacks on neural network circuits, and motivates the work on efficient countermeasures.

3.2. Power2Picture: Using Generative CNNs for Input Recovery of Neural Network Accelerators through Power Side-Channels

The work described in this chapter was published in [4] and is joint work with co-authors Lukas Huegle, Martin Gotthart, Jonas Krautter, Dennis Gnad and Mehdi Tahoori.

The model architecture IP and trained parameters of a Deep Neural Network (DNN) are key assets of the Machine-Learning as a Service (MLaaS) provider and have been widely targeted by side-channel attacks [3], [28], [30], [87]. In contrast, the input data processed by the model is the primary asset of the MLaaS client, especially in privacy-sensitive domains such as healthcare [88]. Despite this, input recovery via side-channel leakage has received significantly less attention.

Existing approaches either require physical access [49] or rely on specific hardware assumptions [29], limiting their applicability to realistic cloud-based MLaaS scenarios.

This work proposes a generic attack based on *generative* Convolutional Neural Networks (CNNs), inspired by Generative Adversarial Network (GAN) models [89]. Instead of generating data from noise, the network is trained to reconstruct inputs from power traces collected via FPGA-internal sensors during inference of a target DNN.

The attack successfully reconstructs MNIST [90] and Fashion-MNIST [91] inputs from FPGA-based accelerators implemented with FINN [31]. It operates in a remote setting and generalizes across different hardware platforms, designs, and operating conditions, demonstrating a realistic threat to MLaaS deployments.

The remainder of this section is taken directly from [4] and is reproduced verbatim.

3.2.1. Attack Methodology

Our attack is essentially a profiling attack based on training a generative machine-learning regression model. We present the general attack flow in Figure 3.9. During an offline profiling phase, voltage estimate traces from a profiling FPGA are used to train the generator model based on the known input images. Afterwards, the attacker can use the trained generator to attack another instance of the neural network accelerator on a multi-tenant victim FPGA, which corresponds to the MLaaS scenario. In the attack phase, collected power traces are fed to the trained model to directly infer the private inputs.

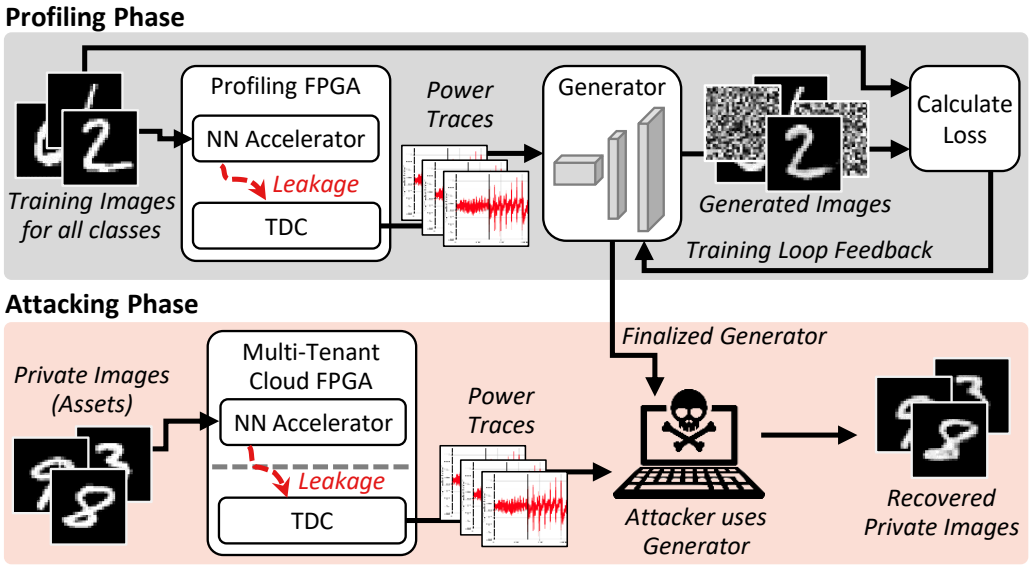


Figure 3.9.: An overview of the entire attack flow, which is based on an offline profiling phase, during which the generator is trained, and an online attacking phase, where the trained generator is used to infer secret input images on a multi-tenant cloud FPGA, where the victim's NN instance performs inference in an MLaaS setup.

3.2.1.1. Image Attacker Generator Network

The rough architecture of our generative CNN employed by the attacker is based around the TensorFlow example on GAN schemes [92], where the generator produces MNIST handwritten numbers from noise inputs.

Instead of using a discriminator loss function for training the generator, we employ a classical regression learning approach. Initially, we train the generator using the classical Mean-Squared-Error (MSE) loss function to reproduce the input images. However, by additionally introducing a term for the difference between image gradients, we are able to significantly improve our input recovery. The final loss function *gradmse* is then defined as follows:

$$\begin{aligned} \text{gradmse}(i_g, i_g) &= \text{MSE}(i_g, i_o) \\ &+ \text{MSE}\left(\frac{\delta i_g}{\delta x}, \frac{\delta i_o}{\delta x}\right) + \text{MSE}\left(\frac{\delta i_g}{\delta y}, \frac{\delta i_o}{\delta y}\right) \end{aligned} \tag{3.1}$$

Here, i_g is the generated output and i_o the original image. The image gradients in x and y direction are denoted as $\frac{\delta i}{\delta x}$ and $\frac{\delta i}{\delta y}$ respectively.

Moreover, we add two fully connected layers of 128 neurons each in the beginning of the generator network, compared to the original GAN example. A summary of the generator architecture is shown in Table 3.3, where we enlist detailed parameters for each layer. During all attacks, we train the generator using the Adam optimizer [93] with a learning

Table 3.3.: Generator architecture with detailed parameters for each layer

Layer	Type	Output features	Kernel	Stride	Padding
1	Linear	128	-	-	-
2	Linear	128	-	-	-
3	Linear	12544	-	-	-
4	Transposed convolution	128	(5, 5)	(1, 1)	(2, 2)
5	Transposed convolution	64	(4, 4)	(2, 2)	(1, 1)
6	Transposed convolution	1	(4, 4)	(2, 2)	(1, 1)

rate of 0.001 and a batch size of 256, which we determined to be optimal during preliminary hyperparameter optimization.

3.2.1.2. Image Victim Network

Exemplary, we attack a low-precision integer-only LeNet [90] model, taken from the quantization aware training library Brevitas [34], which is used in conjunction with the FINN framework [31], that we use to deploy accelerators to Xilinx FPGAs. Our only modification to the original model is a reduction of the input channels to a single channel, as both of our evaluated datasets contain only grayscale images. The activations are quantized to 4 bits, and the bias is quantized to 8 bits.

As the network operates on quantized input images, the accelerator and the measured power traces can only contain information about the quantized inputs. Nevertheless, *training* the generator on the original image data, with values from 0 to 255 may still be beneficial, which is why we evaluate the attack with both quantized and non-quantized images during training. This is usually marked as ‘quant’ and ‘non-quant’ in the results section.

3.2.1.3. Image Recovery Evaluation

To systematically evaluate the proposed attack, we assess the quality of the recovered images using various methods from both previous works on input recovery attacks [29], [49], as well as more generic image processing metrics [94].

Our first metric is the *recognition accuracy*, which has first been proposed in [49] and is supposed to reflect on how well the recovered images would be recognized. To compute the recognition accuracy for a set of recovered images, we reclassify the generated images using the original neural network, which allows to compare the accuracy on the recovered set with the accuracy on the original images.

As an additional evaluation method, we evaluate metrics for image similarity, such as the Mean Structural Similarity Index (MSSIM) proposed by Wang et al. in [94]. The values of the MSSIM range from -1 to +1. A value of +1 indicates that the two images are very

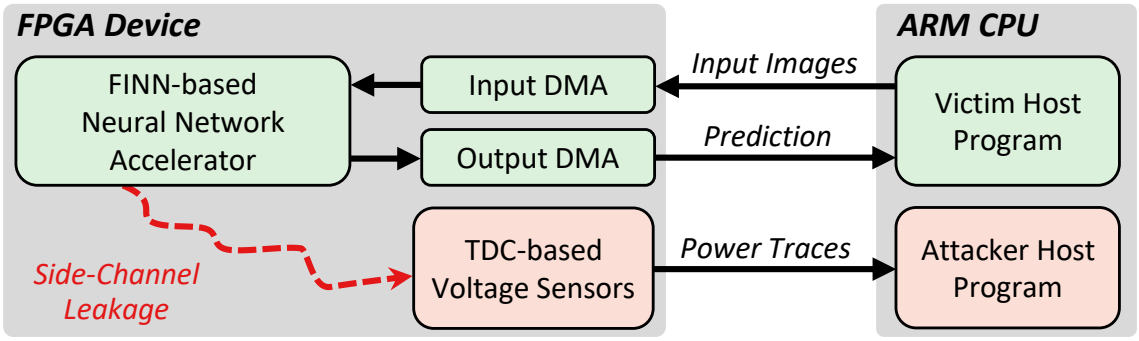


Figure 3.10.: Overview of the implemented setup for both profiling and attack phases in our experiments.

similar or identical, whereas a value of -1 indicates that the images are very different. The MSSIM compares luminance, contrast and structure of the two images. Similar to [29], a sliding-window size of 11 pixels to compute the MSSIM is also used in this work.

A third evaluation metric that we apply is the *pixel-level distance* as proposed in [49]. The pixel-level distance is defined as follows.

$$\alpha_{pixel} = \sum_{x \in I} \frac{||pv(x) - pv_g(x)||_2}{|I|} \tag{3.2}$$

In the above equation, I is the evaluated image, pv is the pixel value of the recovered image, and pv_g is the pixel value of the original image. The evaluated images are more similar the closer the pixel-level distance is to zero.

3.2.2. Experimental Setup

In this work, we use the publicly available framework FINN for generating neural network accelerators for our models. As explained previously, we evaluate attacks on two different datasets, MNIST and Fashion-MNIST, which are split into test (10k samples) and training (60k samples) data as specified in the original publications [91], [95]. After training the corresponding architectures, the networks are exported as ONNX Files, which are further transformed and passed to the FINN compiler. The FINN compiler generates Vivado IP, that can then be deployed as an accelerator onto the FPGA. The basic inference setup is then complemented with sensors for measuring voltage fluctuations. A schematic overview of the overall design is presented in Figure 3.10. Note that we employ multiple Time-to-Digital Converter (TDC) sensors and use the average over all of them for our measurements.

The generated bitstreams are used on two kinds of devices, the Xilinx Pynq-Z1 and the Zynq Ultrascale ZCU104, which are both supported by the FINN compiler. The PYNQ-Z1 features a Z-7020 FPGA SoC with an ARM core and a Xilinx Artix-7 FPGA fabric with 85k logic elements. The Zynq UltraScale+ MPSoC ZCU104 contains 504k logic cells and can fit larger networks. We evaluate our attack mainly on the PYNQ-Z1 platform and only verify that the approach is portable to other platforms on the ZCU104.

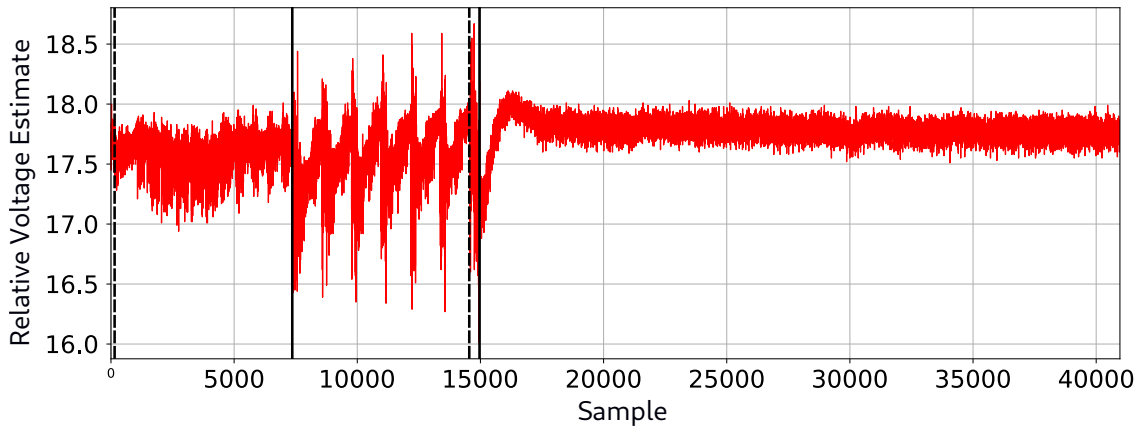


Figure 3.11.: Voltage fluctuation during hardware accelerator execution on the MNIST dataset; average of 100 measurements. The dotted vertical lines denote the time interval during which the 1st layers process data for the first and last time. The solid black lines show the same for the 2nd layers.

The victim model and the attacker's sensors are co-located on the same FPGA. The location of each element is shown in Figure 3.10 as well as the respective data streams between them. The ARM core sends the images to the accelerator via an AXI-bus DMA. The accelerator sends the prediction result back to the ARM core via another AXI-bus DMA. We place five sensors on the board, which collect traces simultaneously. For the training and testing of the generator, either the average of all sensors can be taken, or the sensor closest to the layer of interest can be selected. The TDC sensors clock frequency is twice that of the accelerator. The accelerator runs at 50MHz and the TDCs at 100MHz. We use handshake signals between the DMAs and the accelerator for starting and stopping the data collection. It can be assumed that an adversary is able to recognize the start of the inference as the collected power traces show a visible voltage drop, which can be utilized as a trigger. To interface with our design, we use Jupyter Notebook. Exemplary power traces are shown in Figure 3.11.

To account for a realistic threat model and systematically assess the success of our proposed attack method in different environments, we design a cloud-like MLaaS-scenario on our main experimentation platform, the PYNQ-Z1: The attacker collects traces for training the generator on a local board, without any distinct environmental parameters (room temperature, no EM shielding), which he uses as a device to learn from. Another board is enclosed in an EM shielded climate chamber (Weisstechnik LabEvent T/210/40/EMC [96]), which is remotely accessed and evaluated for different temperature settings. The FPGA design is recompiled, which introduces design variation during the non-deterministic heuristic place-and-route algorithms. This recompiled design is downloaded onto the victim board, which is kept in the temperature-controlled environment. Again, the attacker collects traces, this time on the victim board, which is performing inference for previously unseen images. Finally, we evaluate the input recovery quality on the data collected from the victim board. For comparison, we also show results from attacking a design that has not been recompiled and where traces are collected on the same board.

Table 3.4.: A conclusive overview on the results of all our experiments, where we report recognition accuracy, MSSIM and pixel-level distance for each measurement setup.

Dataset	Platform	Temp. (°C)	Accuracy Baseline	Rec. Acc. quant. / non-quant.	∅ MSSIM (non-quant)	∅ pixel-level distance quant. / non-quant.
MNIST	PYNQ-Z1	25°C	98.7%	96.4% / 95.0%	0.62	32 / 28
MNIST	PYNQ-Z1	0°C	98.7%	96.0% / 95.0%	0.59	28 / 29
MNIST	PYNQ-Z1	70°C	98.7%	94.6% / 94.2%	0.57	34 / 29
MNIST	PYNQ-Z1[1]	25°C	98.9%	92.8% / 92.9%	0.57	34 / 29
Fashion-MNIST	PYNQ-Z1	25°C	88.0%	53.0% / 65.0%	0.38	77 / 82
MNIST	ZCU104	-	98.9%	55.1% / 49.8%	0.49	30 / 34
MNIST	ZCU104[2]	-	98.9%	73.0% / 67.2%	0.60	31 / 34

[1] Retrained victim model with different weights
[2] Attack performed on the same board as profiling

On the larger ZCU104 platform, we verify the general portability of our approach to different architectures, which is why we perform experiments without any temperature control or EM shielding.

3.2.3. Image Recovery Results

We successfully attacked MNIST and Fashion-MNIST inference in various configurations and operating conditions. To evaluate recognition accuracy as introduced in subsubsection 3.2.1.3, we compare with self-trained networks that had the highest accuracy on the respective dataset. For both evaluated datasets we use LeNet for comparison, which has an accuracy of 98.7% on the MNIST dataset and 88.0% on Fashion-MNIST. In Table 3.4 we summarize the results of all attack setups. The first three columns of the table contain information about the measurement configuration, including the used dataset, the hardware platform, and the temperature at which the traces for the attack are collected. We note that the temperature for the training is not defined, as it is performed in a regular office room. The remaining columns of the table show the results of the measurements, including the accuracy of the recovered images, the average MSSIM value, and the average pixel-level distance. In the following subsections, we present and discuss the results for the two architectures in more detail.

3.2.3.1. Input Recovery Attacks on PYNQ-Z1

Here we evaluate the results of attacking both datasets on our main platform, the PYNQ-Z1. Starting with MNIST, we exemplary show the results of image recovery in Figure 3.12. The first row (a) displays the original test images, whereas the second row (b) shows the

a)
b)

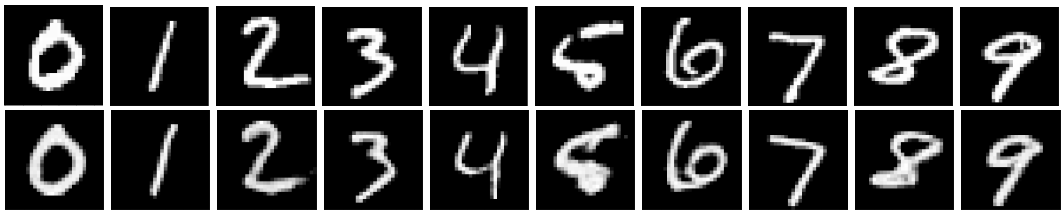


Figure 3.12.: Examples of recovered MNIST images when attacking the LeNet implementation on the PYNQ-Z1. The original images are presented in (a) and the recovered images in (b).

a)
b)

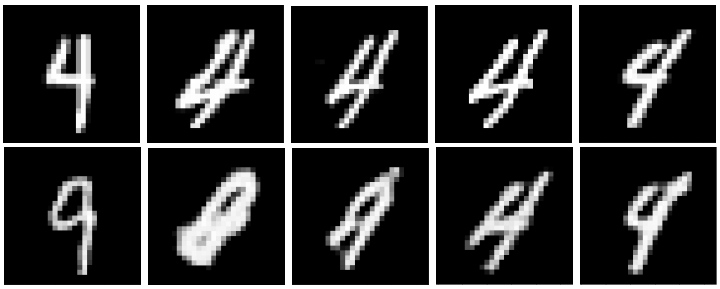


Figure 3.13.: Examples of recovered MNIST images of the number 4, which are misclassified when reclassifying the recovered images with the accelerator on the PYNQ-Z1. The original images are presented in (a) and the recovered images in (b).

recovered images. A simple visual assessment of the recovered images shows that the recovered images are very similar to the original images. The evaluated metrics in the first three rows of Table 3.4 confirm that the attack is indeed successful, as the recognition accuracy is between 94.6% and 96.4% when training the generator on quantized input images, which is almost as high as the accuracy of 98.7% on the original data. The operating temperature of the victim devices has almost no impact on the attack, with the best results observed at 25°C, which is most likely also the closest to the room temperature of the profiling device. Moreover, mean MSSIM and pixel-level distance indicate a high similarity of the original and recovered pictures. The best-case MSSIM value 0.62 is not much worse than, for instance, a JPEG-compressed version of an image, which has a MSSIM of ≈ 0.6949 in the example from [94]. We note, that our mean pixel-level distance is worse than the results presented in [49], but our recognition accuracy is significantly higher, which might be due to the much larger size of our test set (10 000 vs. 200–500). Furthermore, they use a board dedicated for side-channel analysis [97], and they attack the same device they use for training the attack.

We also evaluate the impact of the actual values of the weights used in the victim accelerator at 25°C and present the results in row 4 of Table 3.4. For this setup, we retrain the attacked model with different randomized initial weights, but perform the attack with the generator that has been trained on the originally trained model. Although the evaluation metrics are slightly worse, the recognition accuracy is still over 90%, indicating that the attack works even in scenarios, where the victim only uses a publicly available network architecture with different weights than the attacker.

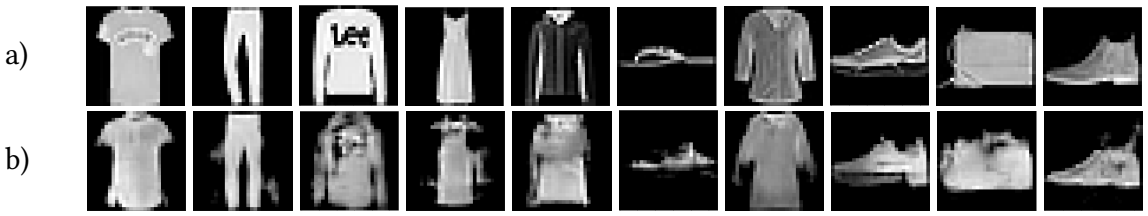


Figure 3.14.: Examples of recovered Fashion-MNIST images when attacking the LeNet implementation on the PYNQ-Z1. The original images are presented in (a) and the recovered images in (b).

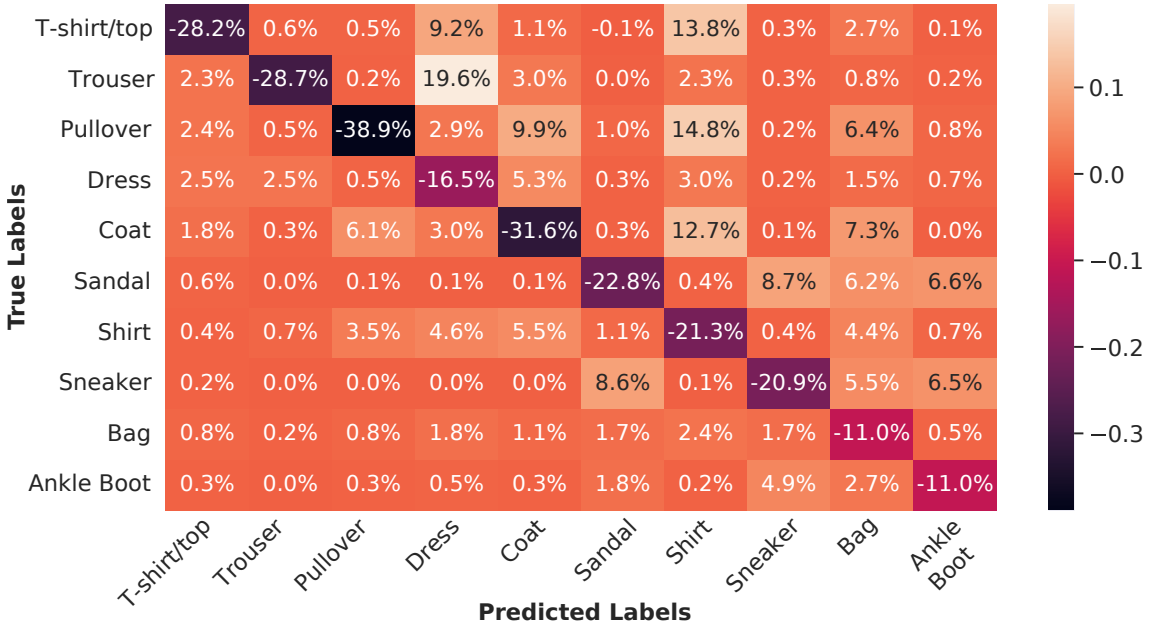


Figure 3.15.: Confusion matrix presenting the accuracy/prediction change when re-classifying recovered Fashion-MNIST images.

When comparing the recognition accuracy of different MNIST classes, we note that the accuracy drop is most pronounced for the class representing the handwritten number four. In Figure 3.13, we show some recovered input images within that class, that are misclassified when evaluated again in the original classifier. We observe, that the images are still quite close to the originals and only small, but critical errors result in the numbers being recognized incorrectly.

Similar, albeit slightly worse results can be observed in Figure 3.14, where we present the original and recovered images that we obtained by attacking a LeNet classifying Fashion-MNIST inputs. As the accuracy on the original test data is only 88.0%, the best recognition accuracy on reclassified images is only 65.0%. Here, the attacker's generator is able to perform better when trained on non-quantized input images, which was not the case on the MNIST dataset. A potential reason could be the greater importance of more fine grained pixel values for the Fashion-MNIST dataset, whereas the handwritten number classification seems to rely more on the information whether a pixel is any other value than zero.

a)

b)

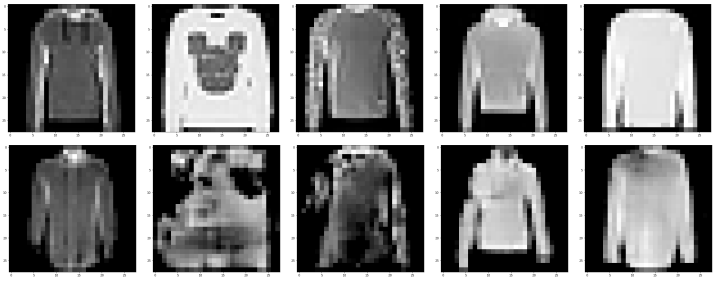


Figure 3.16.: Examples of recovered Fashion-MNIST images of the class 2 (pullovers), which are misclassified when reclassifying the recovered images with the accelerator on the PYNQ-Z1. The original images are presented in (a) and the recovered images in (b).

a)

b)

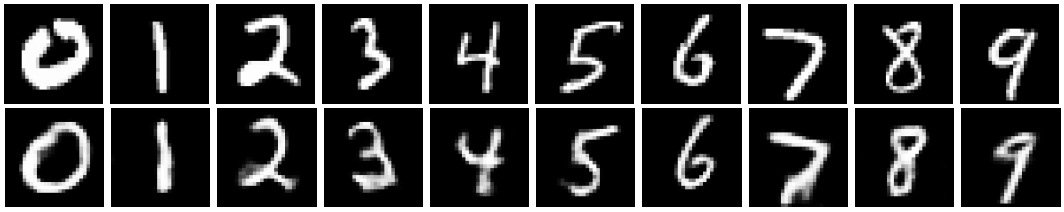


Figure 3.17.: Examples of recovered MNIST images when attacking the LeNet implementation on the ZCU104. The original images are presented in (a) and the recovered images in (b).

For the evaluation of the attack on the Fashion-MNIST dataset, we present a confusion matrix in Figure 3.15, to investigate the accuracy drop during reclassification for each class individually. Here, the class 2 (*pullovers*) has a significant drop of accuracy compared to the original test images. Again we present some misclassified samples of recovered images along with their originals in Figure 3.16. As for the MNIST class of the handwritten number four, we again note a high similarity, where a few small errors cause the misclassification.

3.2.3.2. Input Recovery on the ZCU104 Platform

In the last two rows of Table 3.4, we summarize the results on recovering images from the MNIST dataset on the ZCU104 platform. Examplary, we again show ten successfully recovered MNIST images on the ZCU104 in Figure 3.17, where the similarity is clearly visible. The recognition accuracy in is generally lower than on the PYNQ-Z1 platform, which we assume is due to a much more pronounced power profile on the PYNQ-Z1, where sensor values fluctuate in a much wider range. However, we also see a large difference in recognition accuracy in comparison to an attack on the profiling board, which we present in the last row of Table 3.4.

3.2.4. Section Discussion

Our approach showed good results across two different devices and two different datasets. Compared to previous, highly implementation-tailored attacks, a generator-based image recovery is generic and easy to apply. Moreover, our experiments prove that it is possible to train this generator on a profiling device under different operating conditions and

even with different weights, and attack an established framework for implementing neural networks on Xilinx FPGAs. This scenario corresponds exactly to that of an MLaaS solution, highlighting the threat of such input recovery attacks for clients that process highly sensitive data. Although the proposed approach is highly generic and can be used to attack different accelerators, the parameter space of the generator is also very large and needs to be explored and optimized in future works, to recover inputs from even more complex architectures and datasets.

We presented a novel attack approach where generative neural networks are used to recover inputs to a victim network from power traces that are sampled during inference. The method shows promising results in a realistic threat scenario, where images can be recovered even when training the generator on a different FPGA, with different design variation and different operating conditions. Moreover, we proved its flexibility by attacking different datasets, and different FPGA platforms. Whereas previous works focused on recovering secret IP in the form of neural network structure, parameters, and weights, our work emphasizes the relevance of protecting the privacy of inputs as well, especially in MLaaS schemes. Future research will need to consider if different approaches are required to protect hardware accelerators against both, theft of the provider's IP as well as attacks on the client's privacy.

3.3. Talking Traces: Audio Reconstruction Power Side-Channel Attack

The work described in this chapter was published in [12] and is joint work with co-authors Johannes Reibold and Mehdi Tahoori.

This section presents a power side-channel analysis (SCA) attack that targets the reconstruction of audio inputs processed by deep neural network (DNN) accelerators. In contrast to prior work that mainly focuses on image data, this attack considers speech recognition tasks, where the inputs are audio signals with significantly higher dimensionality and variability.

The proposed attack leverages a generative neural network to reconstruct audio inputs directly from power traces measured during the execution of a DNN accelerator. The attacker model follows a Convolutional Neural Network (CNN)-based architecture similar to an autoencoder and is trained in a supervised regression setting. During a profiling phase, power traces are collected for known inputs and used to train the model to learn the mapping from side-channel leakage to the corresponding audio representation. To further enhance reconstruction quality, a diffusion model is employed as a post-processing step to iteratively refine the generated outputs.

Audio inputs can be represented in multiple forms, which influences both the victim model architecture and the attack complexity. In this work, two common representations are considered, namely raw waveforms and mel-scale spectrograms. Accordingly, two victim

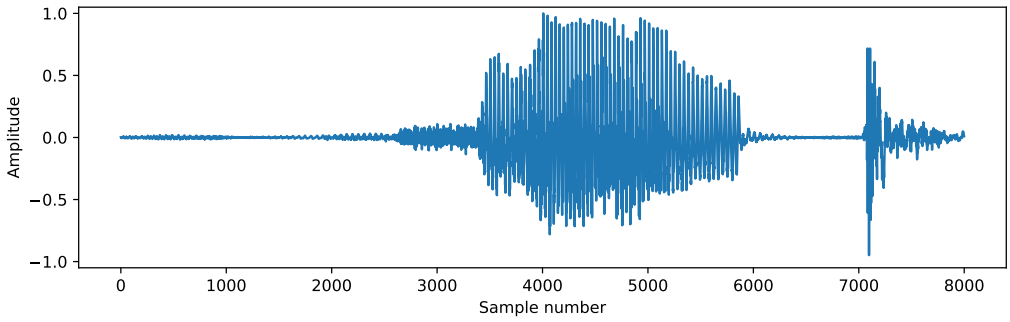


Figure 3.18.: A waveform representing one second of audio at a sample rate of 8 kHz and a bit depth of 16 bit, normalized to $[-1, 1]$.

models are analyzed, each tailored to one of these representations. Compared to prior image-based attacks, these inputs are substantially larger, resulting in either increased parallelism within the accelerator or longer execution traces. Both factors make the side-channel analysis more challenging. Additionally, the evaluation is performed on a dataset with 35 word classes, introducing significantly higher variability than commonly used benchmarks with fewer classes.

The main contributions of this work are summarized as follows:

- First demonstration of a power side-channel-based audio reconstruction attack on realistic DNN accelerators
- CNN-based generative attacker model trained with a deep feature perceptual loss and enhanced using a diffusion-based refinement step
- Reconstruction of both spectrogram and raw waveform representations
- Comparative evaluation of Routing Delay Sensors (RDS) and Time-to-Digital Converters (TDCs) for power trace acquisition

The remainder of this section is taken directly from [12] and is reproduced verbatim.

3.3.1. Audio Representations

Audio signals can be represented in different ways. The main categories of audio representations used for machine learning are raw audio waveforms, spectrograms, acoustic features such as mel frequency cepstral coefficients (MFCCs), latent representations, and symbolic representations such as MIDI or piano rolls [98], [99]. Feature representations are inherently lossy, which is why they are mostly used in older approaches [99]. Latent representations need a neural network, e.g. an autoencoder, to create them in the first place, which needs another representation of the audio signal as an input. Symbolic representations are used for music and are not applicable to speech. This is why waveforms and spectrograms are the most common audio representations in deep learning, especially for applications involving speech.

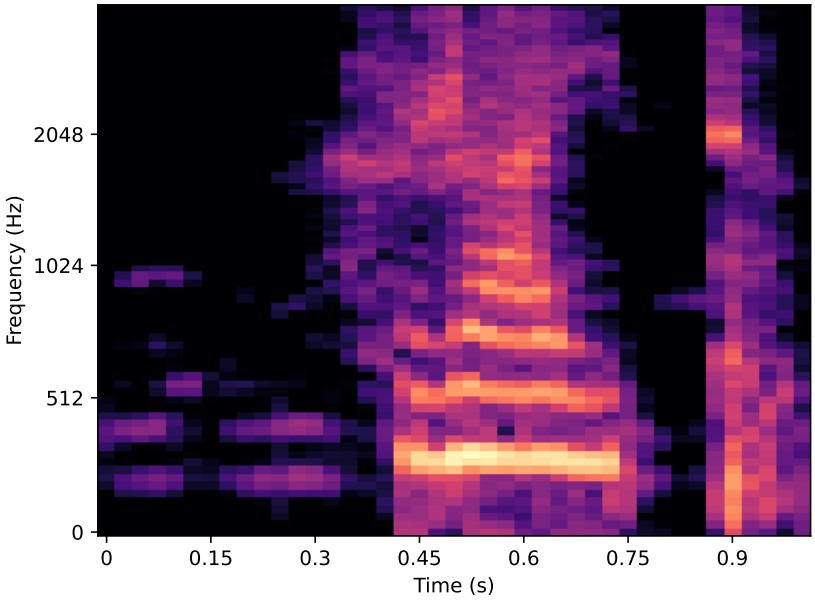


Figure 3.19.: A log-mel spectrogram representing the same audio clip as in Fig. 3.18.

Waveforms are a representation of audio using Pulse Code Modulation (PCM). This means that the amplitude of the continuous audio waveform is sampled in discrete time steps and discrete amplitude values. An example of a waveform can be seen in Figure 3.18. The resolution of the amplitude is defined by the bit depth of the waveform and the size of the time steps is defined by the sample rate.

Spectrograms visualize which frequencies are present in an audio signal at different time steps, rather than representing the underlying waveform directly. The frequency information can be obtained using the Short-Time Fourier Transform (STFT). This produces complex-valued coefficients for each time step and frequency bin, where the magnitude represents the amplitude and the angle represents the phase of the signal components. For spectrograms, only the magnitude is used and the phase information is discarded.

Because it best captures speech information and is often used in the literature both for input representation and loss functions [99], [100], [101], we focus on log-mel spectrograms, shortened to spectrograms for the remainder of this work. These scale both the amplitude values and the frequency bins logarithmically. Figure 3.19 shows an example of a spectrogram.

3.3.2. Neural Networks for Audio Processing

The DNN models in this work are CNNs [90], which are typically used for image applications, but also work for audio and time series data such as power traces. The basic operation used in CNNs is a convolution, which slides a filter / kernel over the input and calculates the scalar product of the kernel and the current section of the input. It is defined by the trainable parameters of the kernel, its size, and the stride with which the operation

slides the kernel over the input. Sometimes, the input is padded with zeros before applying the convolution to achieve a specific output size.

CNNs usually consist of multiple convolutional layers in sequence. Another type of layer used in CNNs is a pooling layer such as max pooling. Many CNNs also use fully connected (FC) layers, which multiply the (flattened one-dimensional) output of a previous layer with a matrix of learnable parameters. FC layers are used, for example, as a classifier at the end of a CNN when it is trained to classify which digit is shown in an input image or which word is spoken in an audio clip.

CNNs can also be used as generative models, e.g. in speech and music synthesis [100], [101], [102], [103]. One generative CNN architecture is the Autoencoder (AE), which consists of an encoder and a decoder. The encoder creates a latent representation of the input with a lower dimensionality, while the decoder reconstructs the input from the latent representation as closely as possible. For waveform or spectrogram inputs, standard 1D- or 2D-CNNs can be used as the encoder model. The decoder uses an inverse of the encoder architecture. It consists of transposed convolutional layers and upsampling (inverse pooling) layers. After training the complete AE, the encoder and decoder models can be used for different purposes. For example, the encoder can be used as a feature extractor to generate the latent representation of input audio, which can then be used for tasks such as classification. The decoder can be used as a generative model to create audio from a given latent representation. An example application of the decoder is the interpolation between two different pieces of music [102].

Diffusion models have recently emerged as powerful generative approaches for audio synthesis. These models operate by gradually adding noise to an input signal and then learning a denoising process that reconstructs clean data from noisy samples. After training, audio can be generated by starting from random noise and iteratively applying the learned denoising steps. Diffusion-based architectures have shown strong performance on tasks such as speech and music generation, particularly when applied to spectrogram representations of audio. By modeling the generative process as a sequence of refinement steps, diffusion models achieve higher fidelity and robustness compared to earlier generative CNNs [104].

3.3.3. Audio Recovery Methodology

To reconstruct the audio signals processed by an FPGA-accelerated victim DNN model, we implement a remote input recovery attack based on generative CNNs. An overview of the attack flow is given in Figure 3.20. The proposed approach is a profiling attack that trains an attacker network in a regression task. Its inputs are power traces containing information from side-channel leakage, which are collected from a pipelined parallel DNN accelerator. The attacker network is then trained using a suitable loss function such that its outputs are similar to the original audio inputs. After profiling, the trained network is used to recover private audio inputs from power traces collected by the adversary in a multi-tenant cloud FPGA setting.

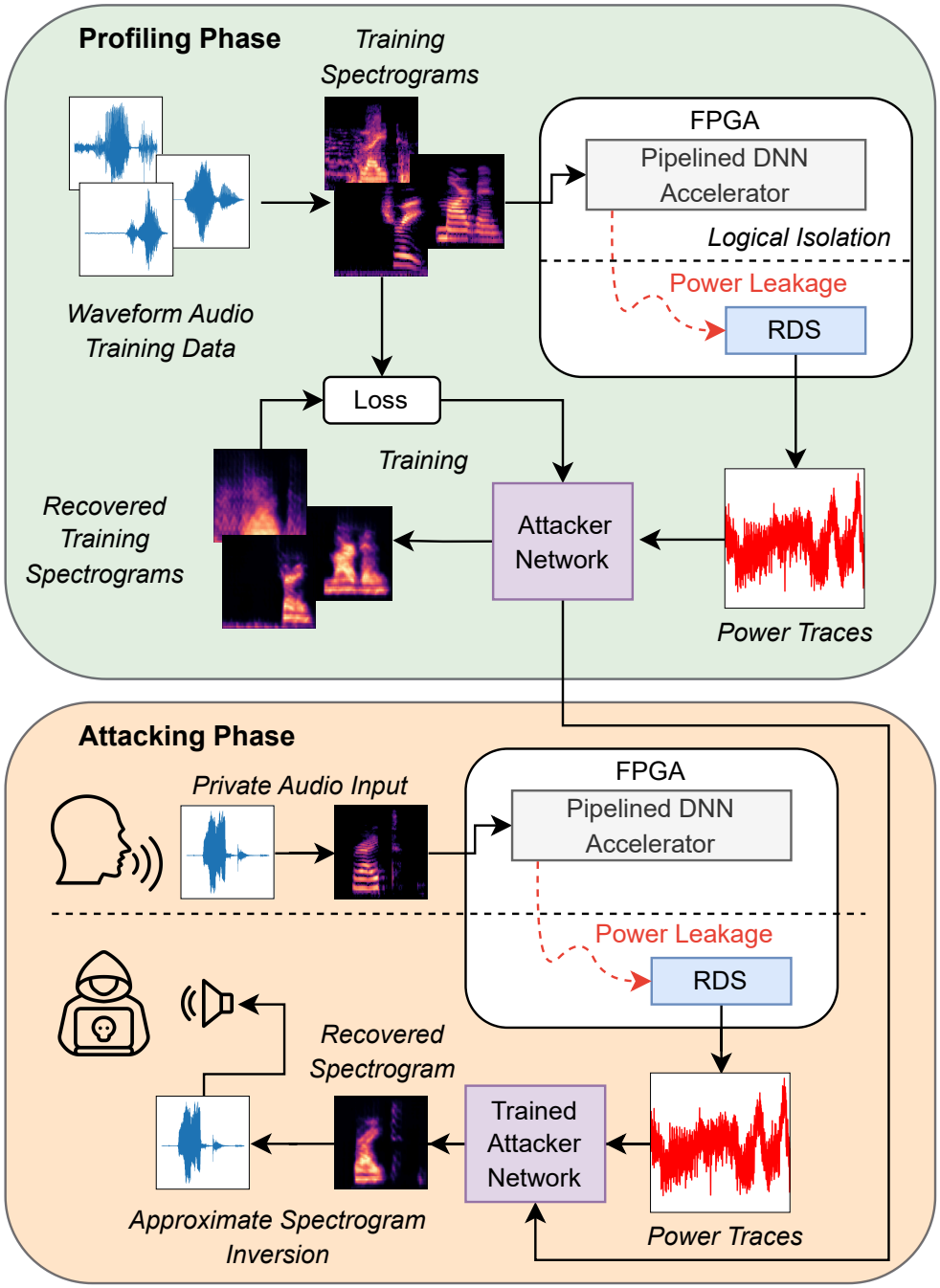


Figure 3.20.: An overview of our profiling side-channel attack on spectrogram inputs. In case of waveform inputs, the conversions from and to spectrograms are omitted and every model works directly on the waveform data.

Our audio reconstruction attack is more difficult than the existing attacks on images for the following reasons:

Larger Input Sizes The input images recovered in most works are from MNIST or similar datasets, which are only 28×28 pixels large. In comparison, the spectrograms used in this work have a size of 80×81 pixels. Raw waveforms are even larger because of the high

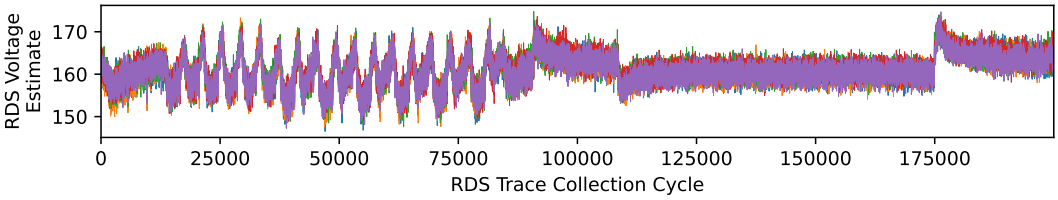


Figure 3.21.: Five example power traces for different accelerator inputs of the spectrogram-based victim model, each shown in a different color. A voltage drop occurs when execution begins at cycle 0, followed by a spike when it finishes around cycle 180000. The first convolutional layer completes around cycle 15000, though this is not easily visible due to overlapping operations in the pipelined accelerator.

sampling rate. Due to the larger input size, the DNN accelerator does more calculations in parallel or takes longer to execute, both of which complicate the input recovery attack. An example of power traces showing this behavior is depicted in Figure 3.21.

Fidelity of Recovered Audio To be able to play back the recovered audio and recognize what has been said requires waveforms and spectrograms with a high fidelity. This is made harder by the fact that spectrograms need to be converted back to waveforms for actual playback, e.g. using the Griffin-Lim algorithm [105]. This is an approximate and lossy step by itself.

Input Variety The audio dataset used in this work contains 35 different word classes compared to the ten classes in MNIST and similar datasets. Additionally, samples from the same class can be very different depending on how the word was pronounced by the speaker and at which time in the audio clip the word was spoken. Due to the higher input variety, the attacker network needs to reconstruct outputs based on fewer similar training examples.

For these reasons, we utilize RDS instead of TDC sensors because of their better sensitivity and find a suitable architecture and training process for the attacker network.

3.3.3.1. Threat Model

Our attack focuses on a multi-tenant cloud FPGA scenario. In this scenario, the FPGA is split into multiple logically isolated regions. Users can only access their regions trough a hypervisor and partial reconfiguration. The adversary is assumed to be a malicious user, who can only deploy a hardware design on their region of the FPGA and there is no physical access possible. Because the whole FPGA uses a shared Power Distribution Network (PDN), the adversary can use FPGA resources to observe power fluctuations caused by the DNN accelerator of a victim user and perform SCA. The start of an inference is clearly distinguishable by the first voltage drop, as can be seen in Figure 3.21, which can be used to align measurements.

We assume the adversary knows the DNN accelerator used by the victim as well as the executed DNN model. This includes its architecture, trained and possibly quantized model

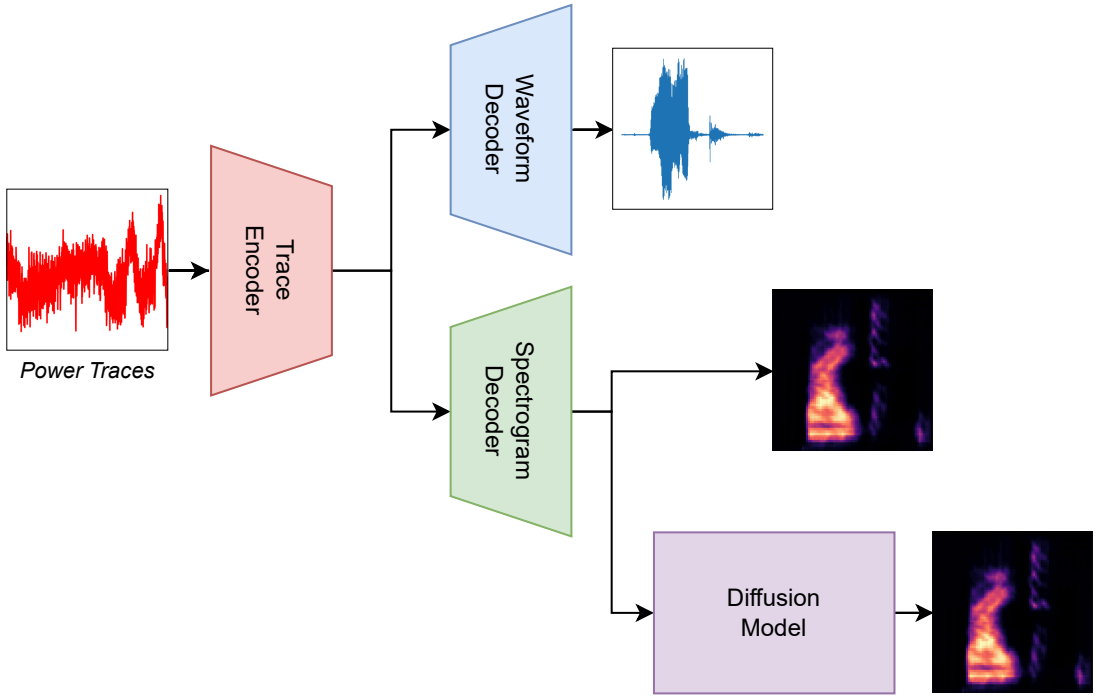


Figure 3.22.: Interplay of the different parts of the attacker networks.

weights and knowledge about the preprocessing of model inputs. This reflects practical deployments in which users rely on publicly available pretrained models as well as open DNN accelerator designs for FPGAs.

3.3.3.2. Attacker Network

The attacker networks used in this work are generative CNNs and have an architecture similar to an autoencoder, consisting of an encoder and a decoder network. The encoder creates a low-dimensional representation of the traces. This representation is given to the decoder to generate the recovered input. In addition to direct spectrogram reconstruction using a generative CNN, we employ a diffusion model as a final refinement stage. Figure 3.22 demonstrates how these submodels work together to create three attacker networks:

- CNN for waveform recovery
- CNN for spectrogram recovery
- CNN with diffusion for spectrogram recovery

We use the trace encoder shown in Figure 3.23a for all attacker networks. To be able to handle large power trace inputs, we use a 1D-CNN architecture. CNN-based architectures have also been shown to work well with large and misaligned power traces for SCA on encryption [106]. Similarly to the 1D-CNNs for waveform inputs [107], we use a large kernel size of 25 and a stride of four in the first layer. Each convolutional (Conv) and fully

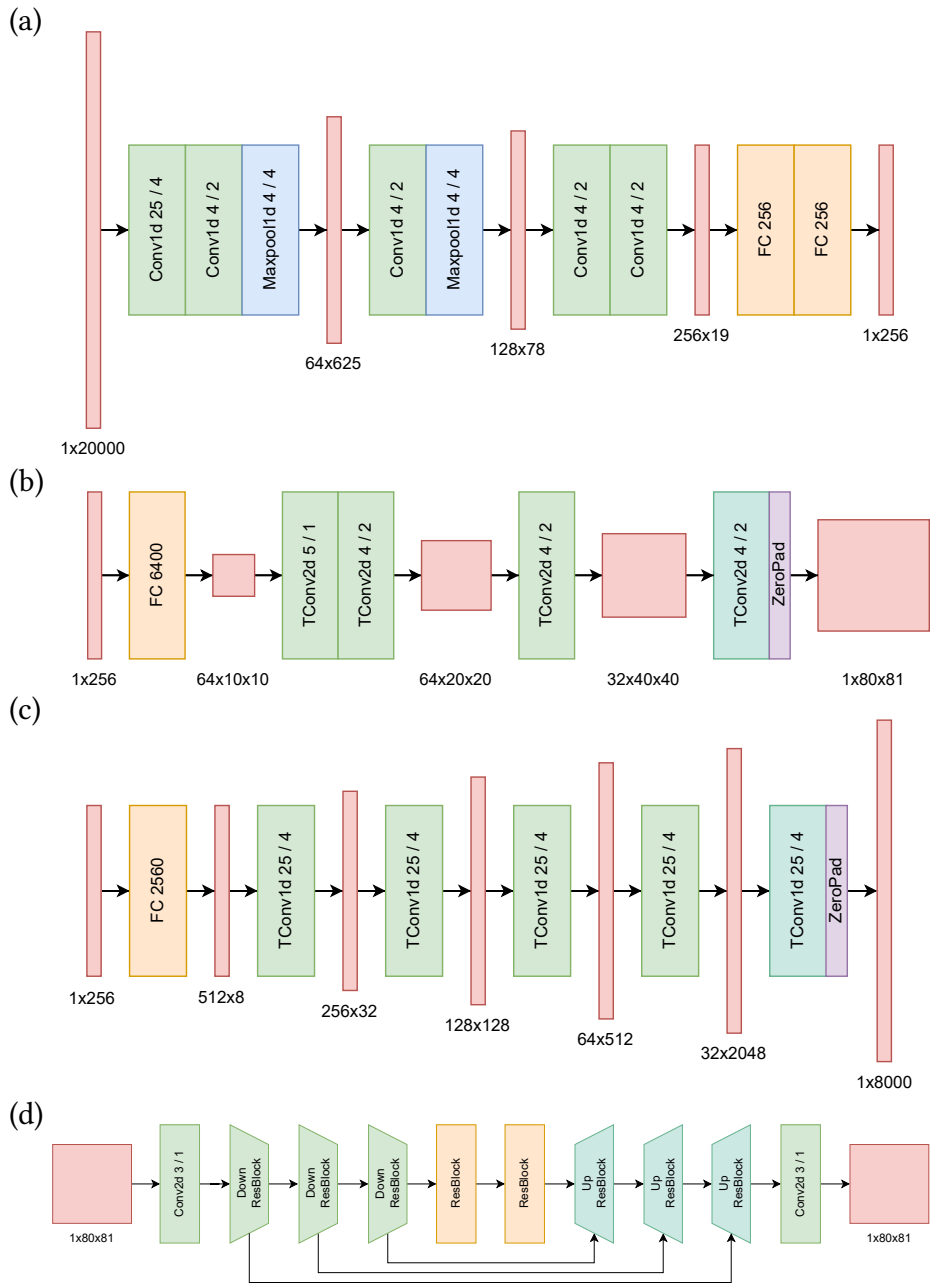


Figure 3.23.: The trace encoder (a), spectrogram decoder (b) and waveform decoder (c) as well as the diffusion model (d) used in the attacker networks. Conv and transposed TConv layers are depicted with their input dimension, kernel size and stride. FC layers include their number of output neurons.

connected (FC) layer in the trace encoder includes batch normalization and a LeakyReLU activation function with a slope of 0.1.

The decoder architectures are taken from existing generative CNNs for images and waveforms, respectively. Our spectrogram decoder is based on the generator model for images used in [108], adapted to the spectrograms used in this work. Figure 3.23b shows its architecture. For the waveform decoder shown in Figure 3.23c, we adapt the WaveGAN ar-

chitecture [103] by using its generator model as a decoder. In both decoders, the transposed convolutional (TConv) and FC layers also include batch normalization and LeakyReLU activation, as in the trace encoder. The only exception are the last TConv layers of each decoder, which omit batch normalization and use the *Sigmoid* activation function for spectrograms and *Tanh* for waveforms.

The diffusion model operates on spectrogram representations and is conditioned on the output of the base attacker model. Instead of generating spectrograms from noise alone, the diffusion model takes a coarse spectrogram estimate as input and progressively denoises it to recover fine-grained temporal and spectral structures. Architecturally, the diffusion model follows a U-Net-style encoder–decoder structure with skip connections, as shown in Figure 3.23d.

Using a suitable loss function to train the attacker network has a high impact on the quality of the recovered inputs. We therefore combine multiple loss functions to achieve the best results. For spectrogram reconstruction, we start with the Mean Squared Error (MSE) between the original spectrograms s_o and reconstructed spectrograms s_r. We also add the MSE between the gradients of the spectrograms as proposed for images in [4]. Finally, a deep feature perceptual loss [109] is added. It uses a classification model M similar to the victim model, but unquantized and with different weights. The deep feature loss function gives the original and reconstructed inputs to the classification model and takes the Mean Absolute Error (MAE) between its intermediate feature maps $M(\text{input}, \text{feature\_map\_name})$. In the case of the CNN attacker network for spectrograms, we use the output feature maps of the fourth convolutional layer. The combined loss function is defined by a weighted sum of the individual loss functions:

$$\begin{aligned} L_{\text{spec}}(s_r, s_o) &= \text{MSE}(s_r, s_o) \\ &\quad + \text{MSE}(\frac{\delta s_r}{\delta x}, \frac{\delta s_o}{\delta x}) + \text{MSE}(\frac{\delta s_r}{\delta y}, \frac{\delta s_o}{\delta y}) \\ &\quad + 0.05 \cdot \text{MAE}(M(s_r, \text{conv4}), M(s_o, \text{conv4})) \end{aligned}$$

The loss weights are chosen based on the following considerations. The normalized spectrogram values are in the interval $[0, 1]$, leading to lower loss values when the error is squared compared to absolute errors. Because of this difference in magnitude, weighting the loss terms equally would make the training process optimize mostly for MAE-based perceptual loss, ignoring the MSE-based terms. In initial training runs, we found a low weight of 0.05 for the perceptual loss results in similar magnitudes for all loss terms and the best reconstruction results. We also briefly experimented with a MSE-based perceptual loss as a solution to this problem, but found it to lead to worse results.

For the combined CNN with diffusion model, the attacker network consists of a base model that produces a coarse spectrogram estimate and a diffusion model that refines this estimate. The base model is trained using the same spectrogram reconstruction loss L_{spec} as defined above. In addition, the final diffusion output is constrained to match the original spectrogram using L_{spec}, ensuring high perceptual and structural fidelity.

The diffusion training objective is defined as

$$\begin{aligned} t \sim \mathcal{U}\{0, \ldots, T-1\}, \quad \epsilon \sim \mathcal{N}(0, I), \\ x_t = \sqrt{\alpha_t}\, s_o + \sqrt{1-\alpha_t}\, \epsilon, \\ L_{\text{diff}} = \text{MSE}(\hat{\epsilon}_\theta(x_t, s_0, t), \epsilon), \end{aligned}$$

where $\hat{\epsilon}_\theta$ denotes the noise predicted by the diffusion model conditioned on the noisy spectrogram x_t, the base model output s_0, and the timestep t sampled from the discrete uniform distribution $\mathcal{U}\{0, \ldots, T-1\}$. The scalars $\alpha_t \in (0, 1)$ define the diffusion noise schedule and control the amount of signal preserved at each timestep. The final training objective for the CNN with diffusion model is given by

$$L_{\text{total}} = L_{\text{spec}}(s_0, s_o) + L_{\text{diff}} + L_{\text{spec}}(s_r, s_o),$$

where s_0 is the base model prediction and s_r is the refined spectrogram obtained from the diffusion model.

For waveforms, we use a multi-resolution STFT (MRSTFT) loss [110] and a deep feature perceptual loss. The first loss uses FFT sizes of 800, 400, and 200 with equally sized window lengths, hop sizes of 200, 200, and 100 and a mel scale with 64 bins. The difference between STFT outputs is measured by the MSE. For the perceptual loss, we use the output feature maps of the third convolutional layer. The combined loss for original and reconstructed waveforms is given by:

$$\begin{aligned} L_{\text{wave}}(w_r, w_o) = 2 \cdot L_{\text{MRSTFT}}(w_r, w_o) \\ + 0.5 \cdot \text{MAE}(M(w_r, \text{conv3}), M(w_o, \text{conv3})) \end{aligned}$$

As with L_{spec}, we choose loss weights based on loss magnitudes and results in initial training runs.

3.3.3.3. Evaluation

Systematically assessing the quality of recovered audio is less explored than evaluating images. One metric we use is the recognition accuracy, which has been used in previous works [4], [49]. This metric generalizes well to different types of input data. It feeds the recovered images to a classification model, in our case the victim model, and calculates the classification accuracy. This correlates well with human perception, similar to the use of deep learning models for deep feature perceptual losses [109].

We evaluate our attack using additional metrics to get more robust results. Quality metrics for audio can be derived from image quality metrics by first creating a spectrogram of the audio. We use the MSE, with errors averaged over all pixels, and Mean Structural Similarity Index Measure (MSSIM) [111] between spectrograms. Both metrics have been shown to correlate with perceived audio quality [112]. The MSSIM values are in the range $[-1, 1]$, with higher values indicating a greater similarity. We use its implementation from [113] with default parameters. For spectrogram recovery, MSE and MSSIM can be

Table 3.5.: Waveform-based victim model

Layer Type	Kernel Size / Stride	Output Features
Conv1d	80 / 16	32
Maxpool1d	4 / 4	32
Conv1d	3 / 1	32
Maxpool1d	4 / 4	32
Conv1d	3 / 1	64
Maxpool1d	4 / 4	64
Conv1d	3 / 1	128
Maxpool1d	4 / 4	128
FC	–	35

Table 3.6.: Spectrogram-based victim model

Layer Type	Kernel Size / Stride / Padding	Output Features
Conv2d	5 / 1 / 2	6
Conv2d	5 / 1 / 2	12
Maxpool2d	2 / 2 / 0	12
Conv2d	5 / 1 / 2	24
Maxpool2d	2 / 2 / 0	24
Conv2d	5 / 1 / 2	32
Dropout 0.5	–	–
FC	–	84
FC	–	35

computed directly, while for recovered waveforms, we first create a 64×41 spectrogram and compute the metrics on these. These spectrograms are smaller than the ones used during the spectrogram recovery because the input waveforms are downsampled. This leads to fewer time steps and a lower maximum frequency contained in the signal due to the Nyquist-Shannon sampling theorem [114].

3.3.4. Experimental Setup

3.3.4.1. Dataset and Victim Models

We use two victim models for waveform and spectrogram inputs respectively. We train them for speech recognition on the Google Speech Commands Dataset v0.02 [115]. It contains 105,829 audio clips with a length of one second each. They are given in audio files as 16 kHz 16 bit waveforms. Every clip contains one of 35 keywords spoken by one of 2,618

Table 3.7.: Training hyperparameters

Network	Optimizer	Batchsize	Epochs	LR / n / γ
Victim				
CNN only	AdamW	512	50	$5{\cdot}10^{-3}$ / 15 / 0.1
Attacker				
CNN only	AdamW	512	50	$1{\cdot}10^{-3}$ / 15 / 0.1
Diffusion		64		$1{\cdot}10^{-3}$ / 15 / 0.2

speakers. The dataset is split into training, validation and test sets, the two latter ones of which contain approximately 10,000 samples each. For the waveform-based model, we downsample the waveforms to 8 kHz. The spectrograms are generated using librosa [116] with the parameters $n_{\text{mels}} = 80$ and $n_{\text{fft}} = 400$, which results in a spectrogram size of 80×81.

We modify the M5 model from [107] for the waveform-based victim model. Table 3.5 contains the full architecture. For the spectrogram-based victim model shown in Table 3.6, we use a modified version of the LeNet architecture [90]. For both models, the convolutional layers include batch normalization an the ReLU activation function. None of the layers uses a bias term, as batch normalization makes it redundant and biases are not compatible with FINN.

The victim models are trained using quantization aware training in Brevitas [34]. All weights and activations are quantized to 4 bits except the input, which uses 8 bit quantization for better classification results. The training hyperparameters are listed in Table 3.7. The learning rate (LR) uses a step scheduler, which reduces the LR after every n epochs by a factor of γ.

3.3.4.2. FINN FPGA Accelerator

We use FINN v0.10.1 [31], [117] to generate FPGA-based accelerators running at a clock frequency of 50 MHz for both victim models. We employ both the AMD PYNQ-Z2 and the larger ZC104 development boards as target platforms. The PYNQ-Z2 features a Zynq-7000 SoC with a dual-core ARM CPU, an FPGA fabric with 85k logic cells and 512 MB of shared DDR3 memory. In comparison, the Ultrascale+ SoC on the ZCU104 features a quad-qore ARM CPU, 504k logic cells and 2 GB of shared DDR4 memory.

The Matrix Vector Activation Units generated by FINN support two weight memory modes, namely *internal_embedded* and *internal_decoupled*. The latter mode is the default and allows for more flexibility, such as changing model weights at runtime without reprogramming the entire design. We enable the decoupled mode for all layers.The streaming of weights in this mode hides some of the side-channel leakage [6], which makes an attack harder and our setup more realistic.

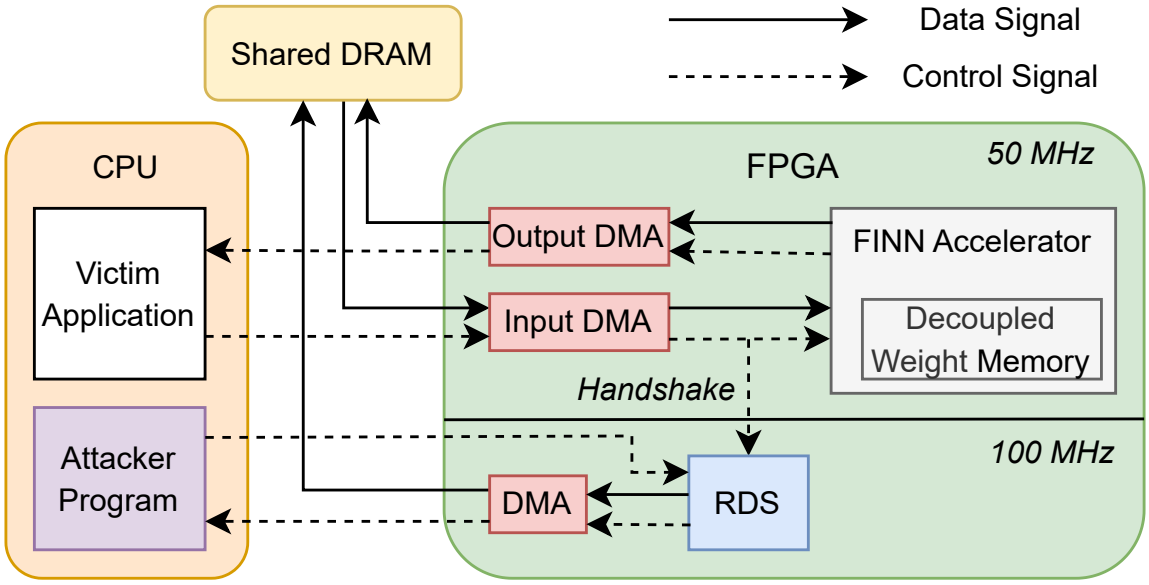


Figure 3.24.: An overview of the experimental setup used for measuring power traces. Data and control signals are transferred using the AXI protocol and using Direct Memory Access (DMA) units.

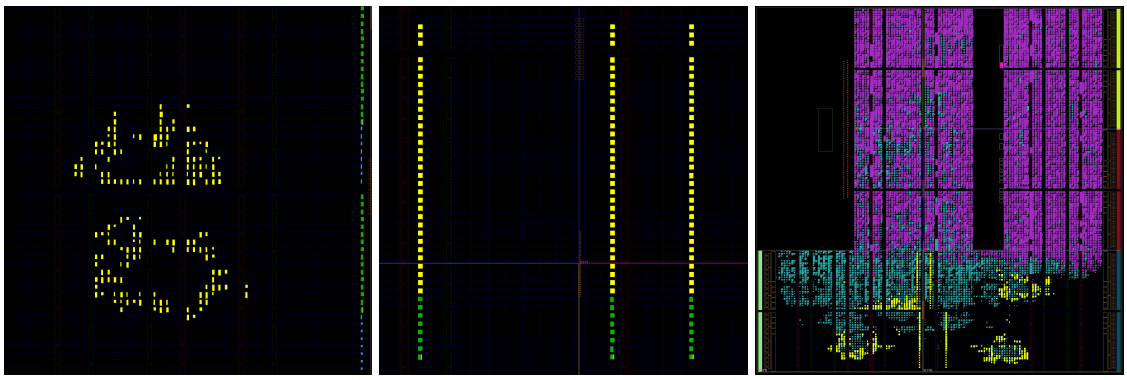


Figure 3.25.: Examples for the placement of (a) two RDS and (b) three TDC sensors on the PYNQ-Z2 board. In (c), both the DNN accelerator (violet) and four RDS sensors (yellow) are placed in different areas of the FPGA.

3.3.4.3. Power Measurement with RDS

We add four RDS sensors running at 100 MHz to the accelerator design generated by FINN. An overview of the measurement setup is given in Figure 3.24. The placement of the accelerator and the sensors is constrained to different areas on the FPGA using PBlocks, achieving logical isolation. This mirrors a multi-tenant FPGA setup. We also create a design with the spectrogram-based accelerator and TDC sensors on the PYNQ-Z2 board to compare the effectiveness of both types of sensors in a profiling attack. Due to the constraint that TDCs must be placed in an FPGA column, they do not fit on the PYNQ-Z2 together with the DNN accelerator. We therefore create a partial accelerator for this comparison, which implements only a part of the spectrogram-based victim model up

to the third convolutional layer. Figure 3.25 shows the placement of the different types of sensors.

To synchronize the power traces, we utilize a handshake signal between the accelerator and the direct memory access (DMA) unit that loads the input data from DRAM. The accelerators exhibit a voltage drop when starting execution and a voltage spike at the end of execution. Some example traces showing this behavior are shown in Figure 3.21. In a real attack, this could be used as a trigger to start collecting power traces. Additionally, we use CNN-based attacker networks, which do not require perfectly aligned power traces [106].

We take the average of the values of the four sensors and collect ten power traces for each audio sample in the dataset. Using multiple power traces per sample, we can compensate for noise in the traces. Either, the ten traces can be averaged, or the attacker network can be trained on all traces and learn to account for noise. Although having multiple power traces available for the same private audio clip is not realistic, using both averaged and non-averaged traces can help evaluate how well the attacker network can account for noise.

3.3.4.4. Attacker Model Hyperparameters

The hyperparameters used for training the attacker networks are listed in Table 3.7. None of the models requires longer than 10 hours to train on an RTX 3080 GPU, while the models without diffusion only require less than 3 hours. For training attacker networks using trace averaging, the ten traces collected for each audio clip are averaged. When training without averaging, each collected trace is used, leading to a 10× larger training set. The traces are normalized to the interval $[0, 1]$ before they are processed by the attacker network.

3.3.5. Audio Recovery Results

We find that our attack is able to accurately recover spectrogram representations of the original audio inputs from power side-channel measurements. In Figure 3.26 we show examples[1] of original and recovered spectrograms for all 35 classes of the speech commands dataset. For these examples, the attacker network that includes a diffusion model for postprocessing was used on power traces from the PYNQ-Z2 board. The recovered spectrograms are close enough to the original spectrograms such that when they are converted to playable waveforms the word can be understood by a human. The evaluated metrics listed in Table 3.8 also show that spectrogram inputs can be reconstructed well. The recognition accuracy is limited by the accuracy of the victim model evaluated on the original spectrograms, which is 92.2%. The lower bound would be a random guess,

[1] When this work is opened in the PDF viewer Adobe Acrobat Reader (Windows / macOS) or Okular (cross-platform), the user can click on the spectrograms in this section. This will play the corresponding sound.

Table 3.8.: Results of spectrogram recovery

Diffusion	Avg.	Recog. Acc. (%)↑	MSE (10^{-5})↓	MSSIM↑
Autoencoder	–	*90.1*	*162*	*0.85*
PYNQ-Z2				
without	yes	78.5	288	0.76
with	yes	81.5 (+3.0)	287	0.80
without	no	75.8	313	0.76
with	no	81.0 (+5.2)	293	0.80
ZCU104				
without	yes	68.9	669	0.73
with	yes	73.8 (+4.9)	385	0.78
without	no	55.9	446	0.73
with	no	62.7 (+6.8)	514	0.76

which in the case of 35 classes is an accuracy of 2.9%. As a further point of comparison, the table also shows the results of an autoencoder. Its architecture consists of only the decoder part of the attacker network plus an encoder that uses the same architecture in reverse. The results of the autoencoder can be viewed as the best results our decoder architecture could achieve under optimal conditions.

On the PYNQ-Z2 board, the recovered spectrograms are very close to the originals with a recognition accuracy of 81.0%. In addition, a low MSE and a high MSSIM are achieved, close to the results of the autoencoder. When comparing the results between averaged and non-averaged power traces, there is almost no difference. This indicates that the attacker network deals with noise in the power traces very well. The spectrograms recovered when attacking the ZCU104 board also exhibit good perceptual quality. However, compared to the PYNQ-Z2 board, all metrics reported in Table 3.8 indicate a slight degradation in reconstruction quality. This trend is also visible in the qualitative examples shown in Figure 3.28. Despite this reduction, the recovered spectrograms remain sufficiently accurate for the spoken words to be clearly recognizable by a human listener.

The attack can also work on audio inputs that contain noise. These are only a small part of the speech commands dataset. Some examples are shown in Figure 3.27. Especially when the noise has a specific frequency or is low volume, e.g. in the examples for “dog” and “yes”, the recovery is not impacted. With noise in the whole frequency spectrum and high volume, the attack cannot recover a recognizable sound. This is the case for the examples “right” and “up”. However, for both examples, the victim model also fails to correctly recognize the word in the original spectrogram.

On waveform inputs, our attack is less successful, but the recovered audio is still recognizable when listening carefully. Some examples of recovered waveforms are shown in Figure 3.28. They are shown as spectrograms, as this is the only representation that can be analyzed and compared visually. The examples show that the attack can produce similar

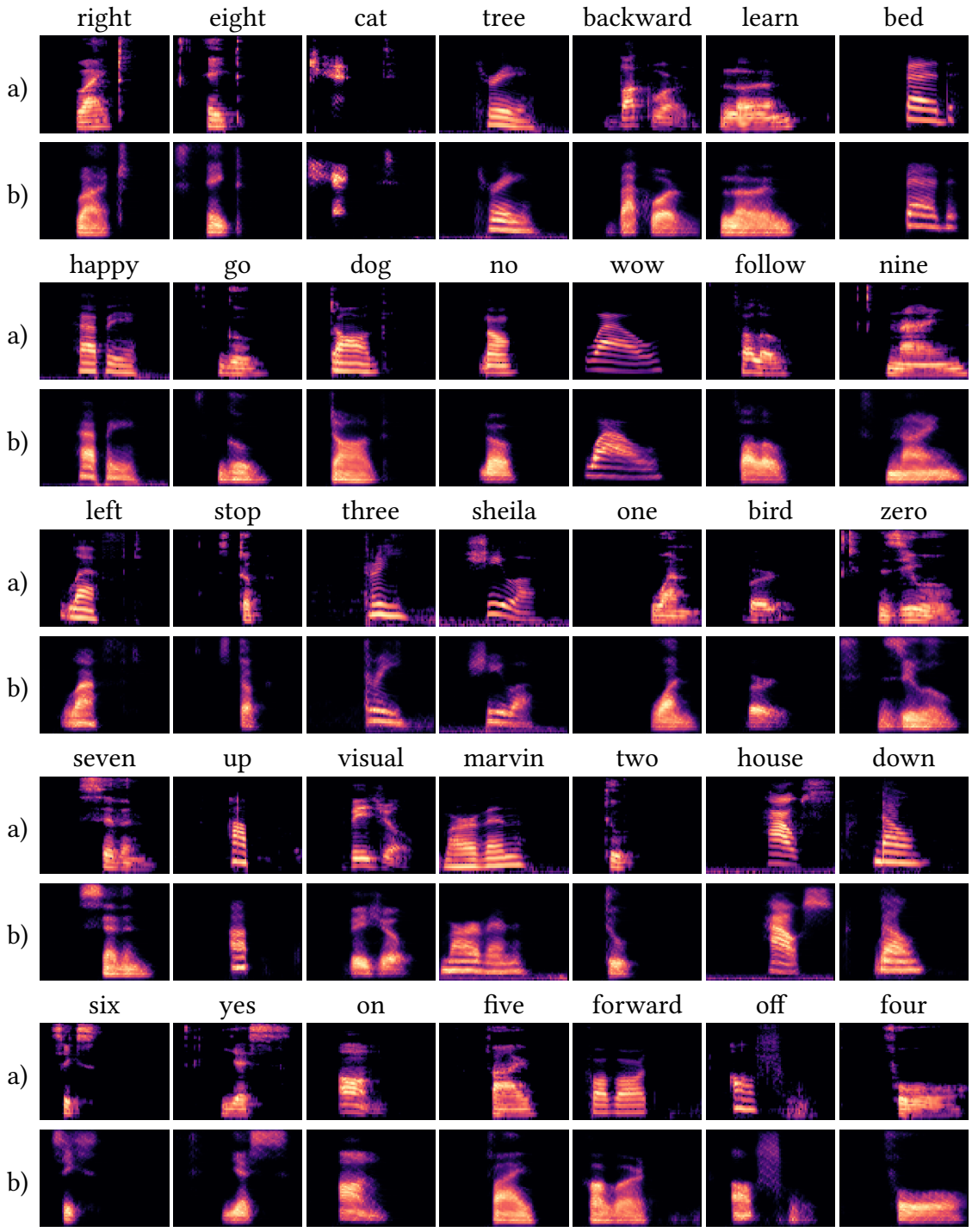


Figure 3.26.: Original (a) and recovered (b) spectrograms on the PYNQ-Z2 board using the attacker network with diffusion

sounds at the right times in the audio clip. This is because amplitude of a waveform can be recovered correctly. However, the individual frequencies contained in the original signal cannot be discerned. The waveforms therefore only moderately sound like the original word. Also the recognition accuracy indicates a lower level of success with a maximum of

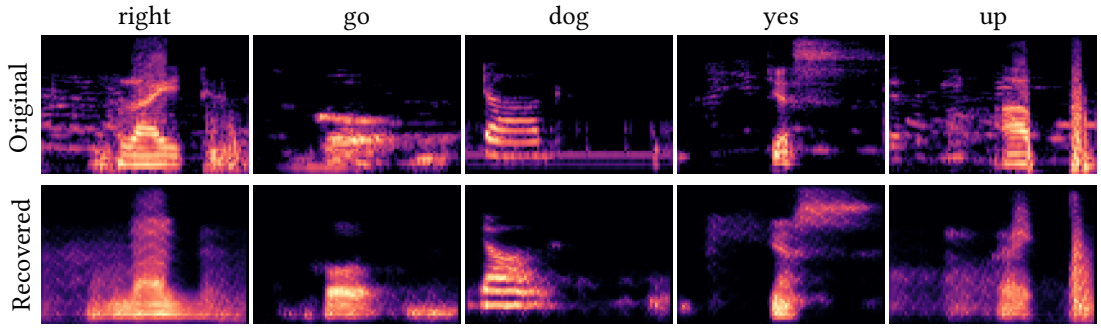


Figure 3.27.: Recovered spectrograms containing noise on the PYNQ-Z2 board using the attacker network with diffusion.

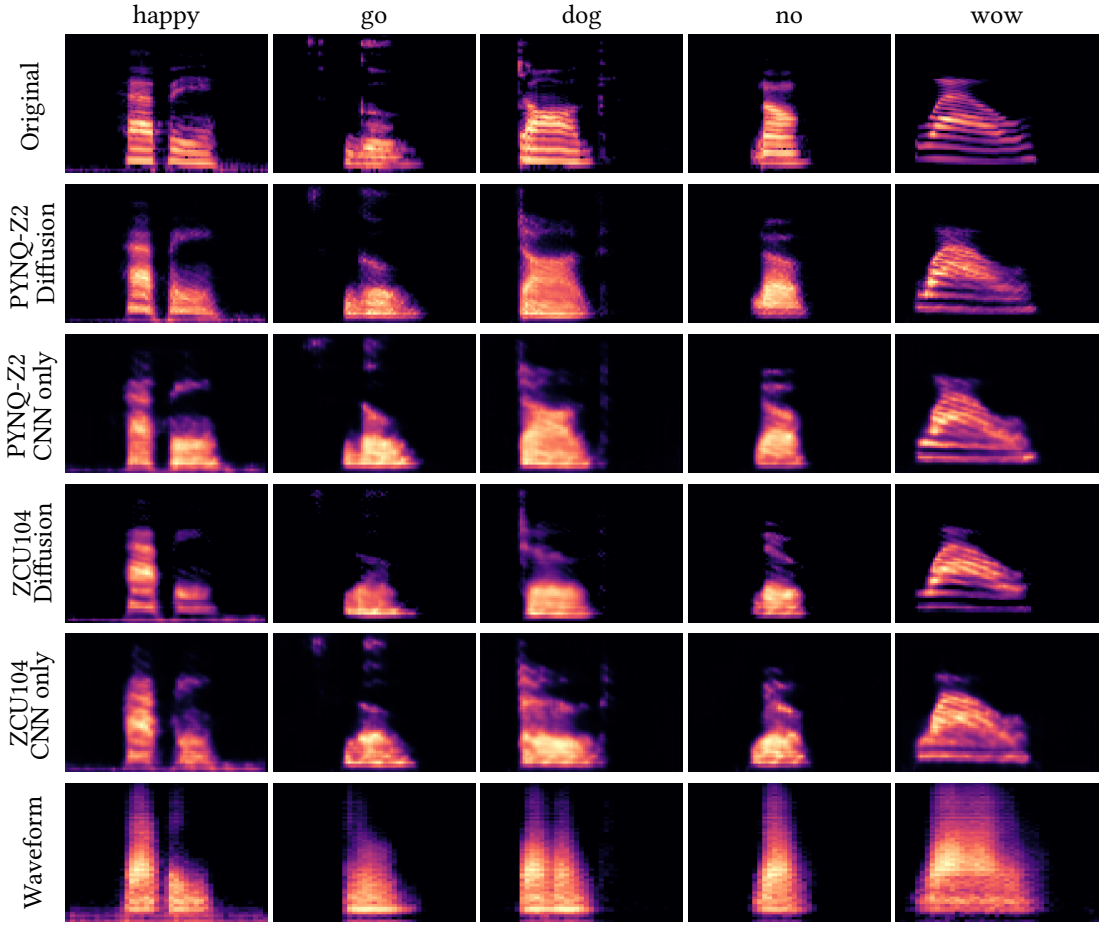


Figure 3.28.: Examples for audio input reconstruction for different input types, FPGA boards and attacker networks.

35.8% on the ZCU104 board. For reference, the accuracy of the waveform-based victim model is 86.1%. Similarly to the results for spectrograms, we also construct a waveform autoencoder for comparison. Although the recognition accuracy of the autoencoder is only 68.5%, its output is of high quality.

Table 3.9.: Sensor Comparison on PYNQ-Z2 for Spectrogram Reconstruction

Sensor Type	Avg.	Recog. Acc. (%)↑	MSE (10^{-5})↓	MSSIM↑
TDC	yes	67.2	358	0.73
RDS	yes	76.3 (+9.1)	296	0.76
TDC	no	60.0	413	0.71
RDS	no	70.5 (+10.5)	346	0.75

3.3.6. Impact of Sensor Type

The power sensor sensitivity has a significant impact on the success of SCA. As existing work on remote SCA mostly uses TDC sensors, we compare them with the RDS sensors used in this work. Table 3.9 shows the attack results for both types of sensors on the partial accelerator using the PYNQ-Z2 board and spectrogram inputs. The RDS sensors perform better in all metrics, validating our choice of sensor. Even with averaged power traces containing less noise, the recognition accuracy is ca. ten percentage points higher when using RDS. These findings align with previous work showing that RDS are more effective for other types of side-channel attack [45].

3.3.7. Attacker Network with Diffusion

We evaluate the spectrogram-based attacker CNN with and without the added diffusion model. The additional refinement of recovered spectrograms using diffusion significantly improves the attack results as shown in Table 3.8. Without the diffusion model, there is a clear difference between the results using non-averaged power traces and the results using less noisy averaged traces. This difference gap is reduced using the diffusion model. Figure 3.28 contains some examples for the attack both including the diffusion model and with the CNN attacker network alone. Especially at the edge features of the spectrogram, the diffusion model achieves a better reconstruction of the original. This leads to a higher clarity of the spoken words during playback.

3.3.8. Practical Challenges

While the proposed attack shows strong results in controlled settings, several practical challenges must be considered in real deployments. Differences between the attacker's profiling setup and the victim system may affect the observed power traces due to the measurement setup, environmental conditions, or manufacturing variations. Additionally, the victim model may not exactly match the profiled model due to fine-tuning or retraining, resulting in slightly different leakage behavior. Multi-tenant environments may also introduce additional noise when multiple workloads execute concurrently on the accelerator. Furthermore, potential countermeasures such as noise injection or randomized execution

may reduce the signal-to-noise ratio, and temporal shifts may require trace alignment. These challenges could be mitigated by collecting profiling traces under varying conditions, applying robust preprocessing and alignment techniques, and training models that are resilient to noise and small model variations.

Another challenge is domain shift, which arises when the victim processes inputs that differ from the attacker's profiling dataset. For example, in speech scenarios the victim user may speak words not included in the training set. To evaluate the generalization capability of our attack in this case, we perform seven additional experiments, each leaving out five of the 35 classes during training, but testing on all classes. On average, the attack achieves an recognition accuracy of 74.42%, with MSE = 307×10^{-5} and MSSIM = 0.80. These results indicate that the attack can generalize to unseen inputs, and larger profiling datasets could further improve this capability.

3.3.9. Section Discussion

This work proposes a novel power side-channel attack to steal private input audio from FPGA-based neural network accelerators. The results of our attack highlights the importance of security and protecting private data as more applications make use of machine learning in potentially untrusted environments. For our experiments, we use FINN as a framework for high performance pipelined neural network accelerators. Our attack uses a generative attacker network, which makes it very flexible and requires minimal knowledge about the accelerator hardware and the structure of the power traces.

The audio spectrograms we are able to recover are larger than the images used in previous works and retain sufficient quality to be clearly recognizable. Furthermore, we demonstrate that the proposed attack scales to a larger number of spoken words while maintaining a good recognition accuracy. These results are enabled by the use of RDS sensors and a generative attacker network employing a CNN-based encoder-decoder architecture. In addition to an extended reconstruction loss with a deep feature perceptual term, we incorporate a diffusion model as a postprocessing stage to further refine the reconstructed spectrograms and recover fine-grained spectral details.

As audio can be represented in multiple ways, we also evaluate the attack on models using raw waveform input. These tests show less satisfactory results, with mostly the location of sounds in an audio clip and the amplitude of the audio signal being recovered. However, in many cases, the word is still recognizable.

4. Training-Induced Side-Channel Amplification

4.1. Leveraging Neural Trojan Side-Channels for Output Exfiltration

The work described in this chapter was published in [2] and is joint work with co-authors Dennis Gnad, Michael Hefenbrock and Mehdi Tahoori.

While prior side-channel attacks on neural network accelerators have primarily focused on recovering model parameters or inputs, the leakage of model outputs has largely been overlooked. However, the outputs of a classifier can themselves be highly privacy-sensitive, particularly in applications such as medical image analysis [118], high-speed vision systems [119], surveillance [120], or speech recognition [121]. In many scenarios, the predicted label is the most valuable piece of information and its disclosure can lead to severe privacy violations. For example, when combined with auxiliary knowledge about a target, extracted predictions may enable targeted fraud, blackmail [122], or personalized exploitation such as targeted advertising [123].

In contrast to input recovery attacks [4], [61], output extraction is more challenging, as the correlation between power consumption and the final prediction is typically weak. To address this limitation, this work introduces a novel attack that leverages neural Trojans to amplify side-channel leakage. Specifically, the training process of the neural network is modified to embed a stealthy Trojan that increases the correlation between power traces and the network output, without affecting its functional behavior.

Building upon prior work [7], this approach repurposes neural Trojans from inducing misclassification to enabling information leakage at the hardware level. Unlike traditional hardware Trojans, the proposed method does not require additional circuitry and thus evades conventional detection techniques such as EM-based analysis [124]. The resulting attack significantly improves output reconstruction accuracy compared to benign models, demonstrating the feasibility of passive output recovery.

The main contributions of this work are as follows:

- Embedding Trojan side-channel mechanisms directly into neural network weights without additional hardware

- Hardware-aware training to amplify correlation between power consumption and classification outputs while preserving functionality
- First demonstration of a passive output recovery attack on neural network accelerators
- Robustness of Trojan functionality under fine-tuning of the model
- Extensive evaluation across architectures, datasets, and operating conditions

The remainder of this section is taken directly from [7] and is reproduced verbatim.

4.1.1. Methodology

In this work, we follow the basic idea of a Trojan side-channel proposed by Lin et al. [125]. However, instead of implementing a dedicated leakage circuit, the Trojan is embedded in the existing logic through the neural network training process. Through malicious training, the neural network itself becomes the side-channel amplifier, without requiring any additional resources or hardware components. This is achieved by manipulating the training process through additional loss terms, enhancing the correlation of the power consumption with the classification results. The outputs of the hardware accelerator can then be recovered through power side-channel analysis. As the Trojan is embedded into the weights of the *PyTorch* (Version 1.13.1) model, it could potentially pose a threat to other hardware accelerators, such as GPUs and TPUs.

Our proposed attack methodology is structured into two distinct phases, as illustrated in Figure 4.1: the Trojan injection during the training phase, and the subsequent recovery of outputs through side-channel analysis. In the initial phase, training was conducted on a GPU to optimize the neural network model, which was then converted into a *PyTorch* representation suitable for deployment. This trained *PyTorch* model was subsequently transformed into Vivado IP, enabling its integration into hardware. The generated IP was then incorporated into our experimental setup for further analysis and testing of the attack's efficacy.

4.1.1.1. Threat Model

Our threat model operates under the assumption of an adversary with access to one or multiple copies of the target device for collecting power traces, which serve as the training data. We assume either the sharing of FPGA devices among multiple users for neural network inference, where potential adversarial actors are co-tenants possessing access to the shared device, or a single device with hidden voltage fluctuation sensors.

The Trojan model is intentionally distributed or shared with potential victims, for instance, through ML IP, allowing its deployment within the victim's hardware infrastructure. The adversary's logic, the voltage fluctuation sensors, are implemented in standard FPGA logic and co-located on the same platform as the victim's accelerator. We assume that the

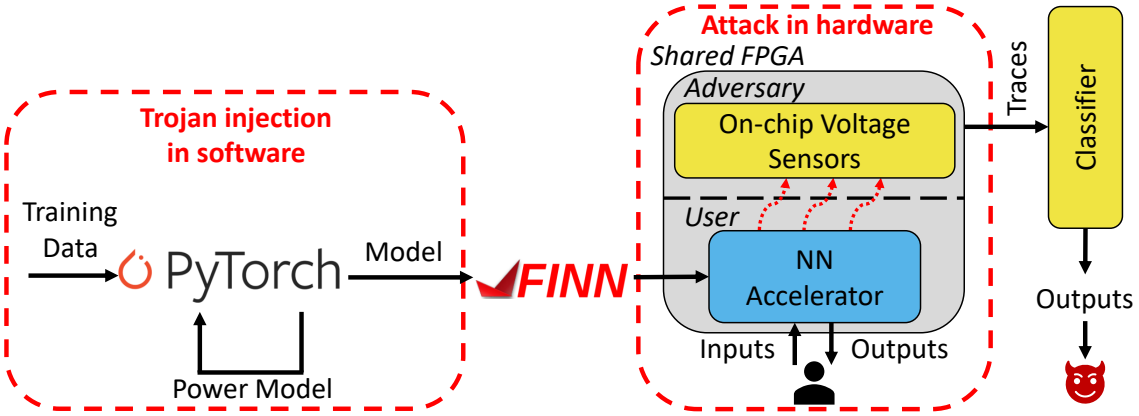


Figure 4.1.: Overview of the attack flow. The Trojan was injected at the training stage. Afterwards, the accelerator was compiled with FINN and deployed on an FPGA platform.

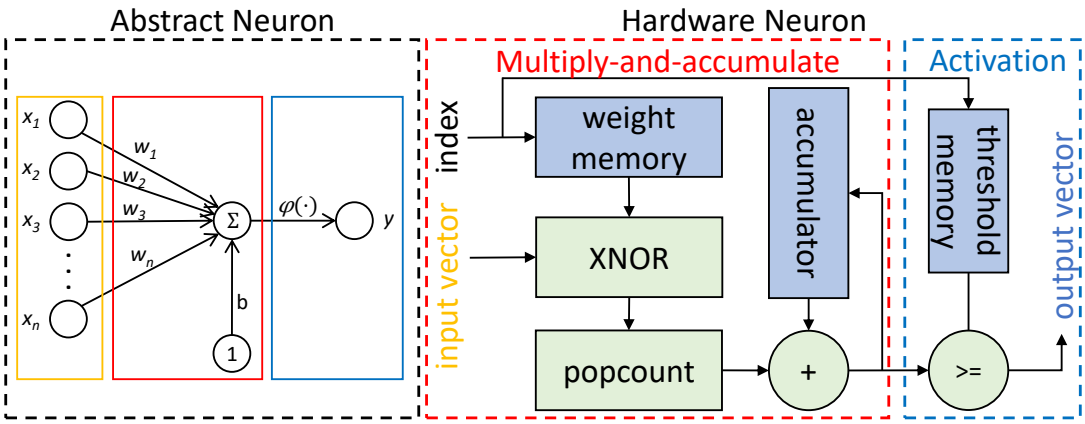


Figure 4.2.: Abstract neuron (left) and hardware implementation of a neuron as in [31] (right).

adversarial logic and the neural network accelerator are logically isolated, such that the only way to access the victim is through the shared power delivery network (PDN).

4.1.1.2. Power Model

In this work, we adapt the loss function to not only optimize the given classification task but also to have a distinct power consumption per output label. To model the estimated power consumption $\mathbf{P}(\mathbf{x}, y)$, we target the MAC operation of hardware neurons. The MAC operation performs a multiplication of the inputs with their corresponding weights and computes the sum of these weighted inputs. The resulting weighted sum serves as the input to the activation function. The estimated power is modeled based on the abstraction of a hardware neuron, as depicted in Figure 4.2, utilizing the weights and quantization levels specified in the model architecture. The power model for a given layer l is represented

by $\mathbf{P}^l(\mathbf{x}, y) := \left[P^l(\mathbf{x}, y)_1, \cdots, P^l(\mathbf{x}, y)_c, \cdots, P^l(\mathbf{x}, y)_C\right] \in \mathbb{R}^C$, where the entries $P^l(\mathbf{x}, y)_c$ are calculated by the sum of bit flips in the XNOR-popcount

$$P^l(\mathbf{x}, y)_c := \sum_{k=1}^{K} \text{Popcount}\left(\text{XNOR}(\mathbf{x}, \boldsymbol{\theta}_k)\right) \cdot \mathbb{1}_{\{y=c\}}. \tag{4.1}$$

Here , K is the number of neurons in layer l, c denotes the output index encoding the class, and $\boldsymbol{\theta}_k$ the corresponding parameters of the neuron. Finally, $\mathbb{1}_{\{\cdot\}}$ denotes the indicator function returning 1 if the respective condition is met or else 0.

Using *PyTorch*, the power consumption for a layer is simulated by applying the ReLU function to the layer's weighted input and subsequently taking the sum. The returned value represents the switching activity resulting from the weight multiplication. Additionally, we can model the popcount by taking the sum of '1's in the weighted sum's binary representation. As the conversion to a binary representation is not differentiable, a straight-through gradient estimator [126], which ignores said conversion, is applied.

4.1.1.3. Trojan Injection Through Training

In our approach, we make our training hardware-aware and target the MAC operation, which has been shown to be the most vulnerable part of a neural network accelerator [127].

For this purpose, we aim to increase the information leakage from the profile of the power consumption. More specifically, we try to enforce distinct patterns in the power consumption for each class. Ideally, the resulting patterns vary little for examples $\mathbf{x}$ of the same class, but greatly for examples of different classes. This can be formalized as follows. Let $\overline{\mathbf{P}^l} := M^{-1} \sum_{m=1}^{M} \mathbf{P}^l(\mathbf{x}_m^{(l-1)}, y_m)$ denote the vector of the average power consumptions over a subset of the data of size M (N in the case of the entire dataset) of layer l. Here, $\mathbf{x}^{(l-1)}$ is the output of the previous layer. Consequently, $\mathbf{x}^{(0)} = \mathbf{x}$. To encourage high similarity in the patterns of the power consumption for inputs $\mathbf{x}$ of the same class, we define

$$\lambda_{\text{pull}}^{M}(\mathbf{x}, y) := \sum_{l=1}^{L} \left\| \mathbf{P}^l(\mathbf{x}^{(l-1)}, y) - \overline{\mathbf{P}^l} \right\|_2^2. \tag{4.2}$$

Through $\lambda_{\text{pull}}(\mathbf{x}, y)$, the power patterns are pulled towards their mean, which should ensure high similarity. Additionally, the dissimilarity between patterns of different classes can be assessed through

$$\lambda_{\text{push}}^{M}(\mathbf{x}, y) := \sum_{l=1}^{L} \sum_{k=1}^{L} \left\| \overline{\mathbf{P}^k} - \overline{\mathbf{P}^l} \right\|_2^2, \tag{4.3}$$

which encourages large distances between the means of the patterns in the power traces for different classes. Note that $\lambda_{\text{push}}^{M}$ depends on M, which indicates the (number of) samples used to calculate the means $\overline{\mathbf{P}^l}$. Finally, we introduce

$$\lambda_{\text{corr}}^{M}(\mathbf{x}, y) := 1 - \mathbf{r}_{p,y}^{\top} \, \mathbf{R}_{p,p}^{-1} \, \mathbf{r}_{p,y}, \tag{4.4}$$

with $\mathbf{r}_{p,y}$ denoting the vector of Pearson correlation coefficients between y and the $\mathbf{P}^l(\mathbf{x}, y)$ for $l = 1, \cdots, L$. Here, $\mathbf{R}_{p,p}$ denotes the correlation matrix between the power models of different layers l. Having a high correlation should increase the information contained in the power trace for each class. This is because the correlation directly determines the predictive performance of a linear discrimination function. Note that, similarly to λ^M_{push}, $\lambda^M_{\text{corr}}(\mathbf{x}, y)$ also depends on the number of data points M used to estimate the means $\overline{\mathbf{P}^l}$ for the layers.

We combine these terms with the function

$$\text{Leak}(\mathbf{x}, y) := \alpha_{\text{pull}}\lambda^M_{\text{pull}}(\mathbf{x}, y) - \alpha_{\text{push}}\lambda^M_{\text{push}}(\mathbf{x}, y) + \alpha_{\text{corr}}\lambda^M_{\text{corr}}(\mathbf{x}, y) \tag{4.5}$$

to encourage information leakage through the power trace. The preceding scalar coefficients $\alpha_{\text{pull}}, \alpha_{\text{push}}, \alpha_{\text{corr}} \in \mathbb{R}^+$ denote tuning parameters. Note that, due to common convention regarding neural network training as a minimization problem, $\text{Leakage}(\mathbf{x}, \mathbf{y})$ is defined such that low values are favorable. The term λ^N_{push} is thus inverted to encourage high dissimilarity, and the negative correlation is used in Equation (4.4).

Consequently, our training objective becomes

$$\min_{\boldsymbol{\theta}} \quad \sum_{n=1}^{N} \text{Loss}\left(\mathbf{f}_{\boldsymbol{\theta}}\left(\mathbf{x}_n\right), y_n\right) + \text{Leak}(\mathbf{x}_n, y_n). \tag{4.6}$$

Since all terms involved are differentiable with respect to $\boldsymbol{\theta}$ (almost everywhere), the training objective qualifies for classic gradient-based optimization. Additionally, stochastic versions can also be applied, where unbiased estimates of the gradient are obtained from samples (mini batches) of size M of the entire dataset. Note that this influences the estimation of $\lambda_{\text{pull}}(\mathbf{x}_n)^M$ and λ^N_{corr}, which depend on the size of the mini batch M. Generally, larger mini batches are preferable, as they lead to more stable gradient estimates.

4.1.1.4. Output Recovery

We assume that the adversary trains the neural network and subsequently either distributes the accelerator or offers it as a service accessible to users. In the context of the attack scenario, we adopt methodologies established in prior research on passive-voltage-based side-channel attacks in multi-tenant FPGA environments [27]. In this setup, the neural network and the voltage sensing logic are logically isolated. Additionally, the adversary is limited to passive observation of the power consumption and does not have direct control over the inputs provided to the network. Despite this limitation, the adversary attempts to recover the network's output solely from the measurements collected during inference.

This recovery is accomplished by training a classifier using the observed power traces and their corresponding classification results. During this training phase, the adversary is assumed to have control over the inputs to the network, enabling the collection of labeled power traces to construct the classifier.

We make the following assumptions:

- Inputs to the network are not controlled and are random
- Start and end time of each individual inference is known

4.1.2. Experimental Setup

To evaluate the proposed approach, we conducted experiments using two Zynq UltraScale+ MPSoC ZCU104 boards, a Xilinx platform equipped with approximately 504k logic cells. This platform is compatible with the FINN framework, an open-source tool developed by Xilinx for compiling and deploying quantized neural network accelerators derived from *PyTorch* models in the ONNX format [31]. One of the boards was placed in a shielded climate chamber (Weisstechnik LabEvent T/210/40/EMC) to facilitate controlled experiments under varying temperature conditions.

The FINN framework provides support for time-multiplexing of computations through neuron and synapse folding. This enables efficient deployment of complex models, while maintaining high performance and minimal resource usage. For our experiments, all neural network models were quantized to 1-bit weights and activations using the *PyTorch* library *Brevitas* [34], which incorporates quantization-aware training. This setup ensured that the models were both resource-efficient and well-suited for hardware deployment, enabling comprehensive analysis of the Trojan's impact in realistic scenarios.

4.1.2.1. Neural Network Models

In the following, we denote MLP-X as a binary MLP with X neurons in all of its hidden layers. All MLPs consisted of 4 layers and were trained on the MNIST dataset. The MLP-64 was fully unfolded, while for the MLP-128, we applied a neuron fold of 2 in all layers, and for the MLP-256, a neuron fold of 4. The CNN had a VGG-like architecture [128], which consisted of 2 convolutional layers with 64 output channels, 2 with 128 output channels, and 2 more with 256 output channels, followed by two fully connected layers with 512 neurons, and finally an output layer with 10 neurons. For the CNN, only the first 4 convolutional layers were unfolded, and the subsequent layers had some degree of folding. We injected the Trojan into the CNN in the first four layers and left the remaining layers untouched. The VGG-like model was trained on the CIFAR-10 dataset [129] and BloodMNIST [130]. Furthermore, we evaluated the Trojan for a subset of the *Speech Commands* [115] dataset, for which we trained a 3-layer MLP-256, here denoted as KWS-256. Our subset consisted of 11 classes: *unknown*, *down*, *go*, *left*, *no*, *off*, *on*, *right*, *stop*, *up*, *yes*. All class labels from the original dataset, which are not in this list, are labeled *unknown*.

In Table 4.1, we show the hyperparameters chosen for the training of the different Trojan networks. These parameters were decided by a grid search. A batch size of 1024 proved to be optimal for the training. The accuracy scores on the respective test sets for the benign and Trojan networks are also presented. Our Trojan only slightly impacted the accuracy of

Table 4.1.: Prediction accuracy and hyperparameters α_{pull}, α_{push}, α_{corr} for training on the respective datasets.

Network	Dataset	Test Accuracy		Hyperparameters		
		Benign	Trojan	α_{pull}	α_{push}	α_{corr}
MLP-64	MNIST	91%	91%	0.0005	0.001	0.1
MLP-128	MNIST	96%	93%	0.0005	0.001	0.1
MLP-256	MNIST	97%	95%	$1 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	0.1
KWS-256	SpeechCommands	88%	87%	0.0005	0.001	0.1
VGG	CIFAR-10	82%	78%	$1 \cdot 10^{-7}$	$5 \cdot 10^{-8}$	0.1
VGG	BloodMNIST	92%	91%	$1 \cdot 10^{-7}$	$1 \cdot 10^{-6}$	0.3

the NN. In the worst case, which was for the VGG (CIFAR), the accuracy score dropped by approximately 4%, while for the other networks, it was around 2%. Generally, the training time was slightly longer than benign training, due to the added constraints for embedding the Trojan; however, all calculations could be efficiently performed on a GPU.

4.1.2.2. Sensor Setup

Both the RDSs and the neural network accelerators were implemented on the FPGAs, without any direct connection between the attacker and the victim logic. A floor plan for a VGG (BloodMNIST) is presented in Figure 4.3, with the sensors colored in red and the layers of the network in different colors. The accelerators ran at 50 MHz and the RDSs at 100 MHz. We chose 50 MHz for the accelerators, to have a common setup among the different models. With correct calibration, it is also possible to run some of the accelerators and the RDSs at higher frequencies, thus our attack is not limited to these values. In the original RDS paper [45], the sensor and target circuit frequency were set to 200 MHz and 20 MHz. By setting a lower frequency for RDSs, we reduced the number of collected samples per trace, and thus shortened the duration and lowered the complexity of the attack.

Voltage fluctuations were recorded into on-chip BRAM, which were later transmitted to the ARM core via AXI bus. We applied 0 to 10× averaging for the power trace acquisition and compared the results. For this purpose, inferences were repeated a certain number of times and the average of all traces was calculated.

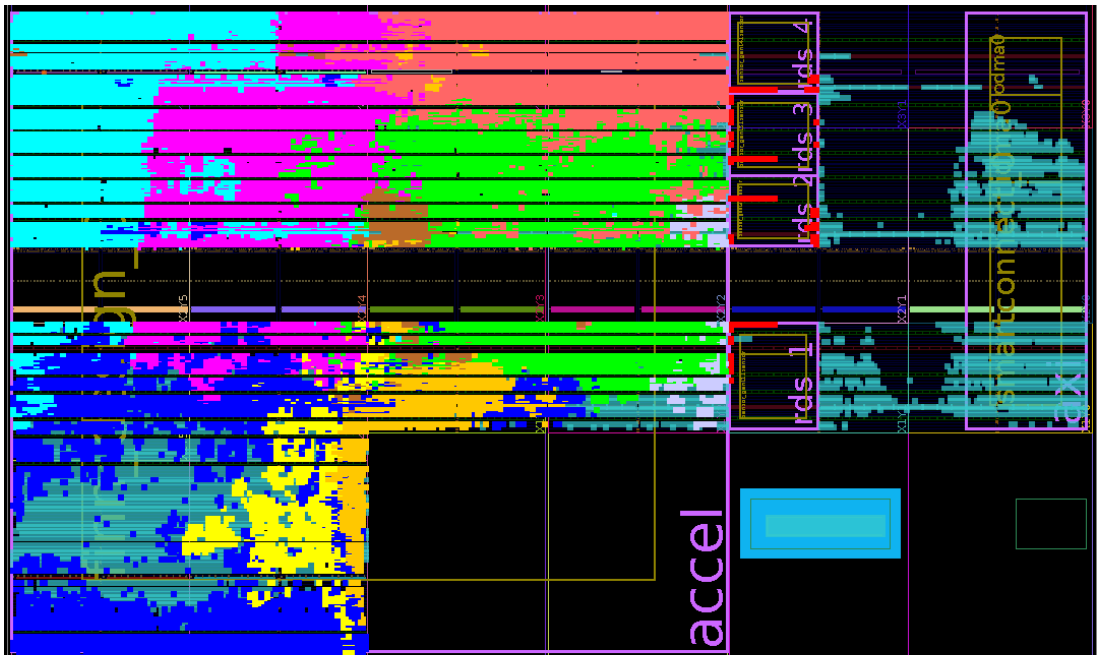


Figure 4.3.: Exemplary floor plan of a CNN mapped onto the FPGA. Layers: (1) **Yellow**, (2) **Violet**, (3) **Green**, (4) **Pink**, (5) **Blue**, (6) **Light blue**, (7) **Orange**, (8) **Brown**; Output layer: **Lilac**. RDSs: **Red**; Control logic: **Cyan**.

4.1.2.3. Output Classification Setup

For the classification of power traces to outputs, we used *Auto-Sklearn 2.0*. This AutoML system reduces the user effort by carrying out optimization across preprocessors, classifiers, regressors, and their hyperparameter settings collectively. The classifiers were trained on power traces from both devices, including temperature variations. The setup is presented Figure 4.4. Inputs were sent from the ARM-core to the FPGA, which ran the accelerator and returned the outputs, while the voltage measurements were performed with the on-chip sensors. As we assumed that the attacker could observe outputs during the training, there was no distinction between the profiling and target device. For data acquisition, we fed the accelerator with images from the test sets (10 k images) of the corresponding datasets and also saved the network's prediction for the purpose of training the classifiers. Traces were split into 80% training and 20% test samples. During the attack, the adversary had no control over either of the devices and could only passively measure voltage fluctuations.

4.1.3. Results

As a preliminary experiment, we analyzed the effect of our training on the power estimate from the MAC operation of the benign models versus the manipulated networks. During the training, we observed the correlation of the power estimate of the network with the

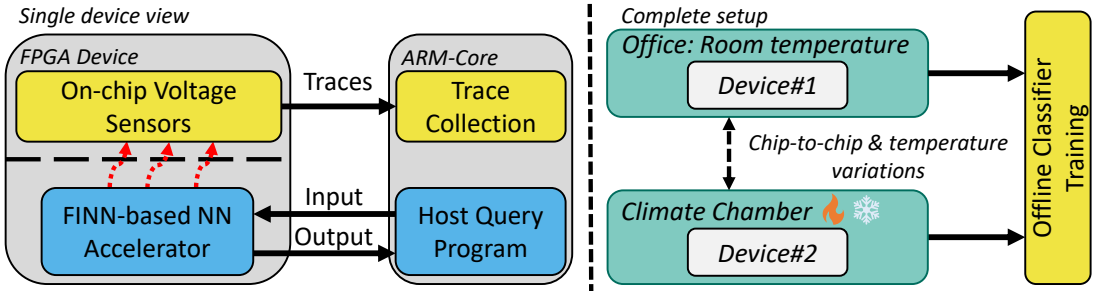


Figure 4.4.: Overview of the setup for a single device (**left**) and the complete setup (**right**) with two devices.

predicted class label for each prediction. In Figure 4.5, it can be seen that the correlation for benign models was around 0.25 lower than their Trojan counterpart. The benign CIFAR-10 was an extreme case, with a value of only 0.22, which was increased to 0.63 by the Trojan. For the attacker, who compares the benign and Trojan model MAC-output correlations, this can be used as an indicator of successful output extraction on the actual hardware. By increasing the parameters α_{pull}, α_{push}, and α_{corr}, the correlation can be strengthened at the cost of classification accuracy. For the VGG-like models, we delayed the activation of the malicious training until after the first 10 epochs to prioritize improving the prediction accuracy.

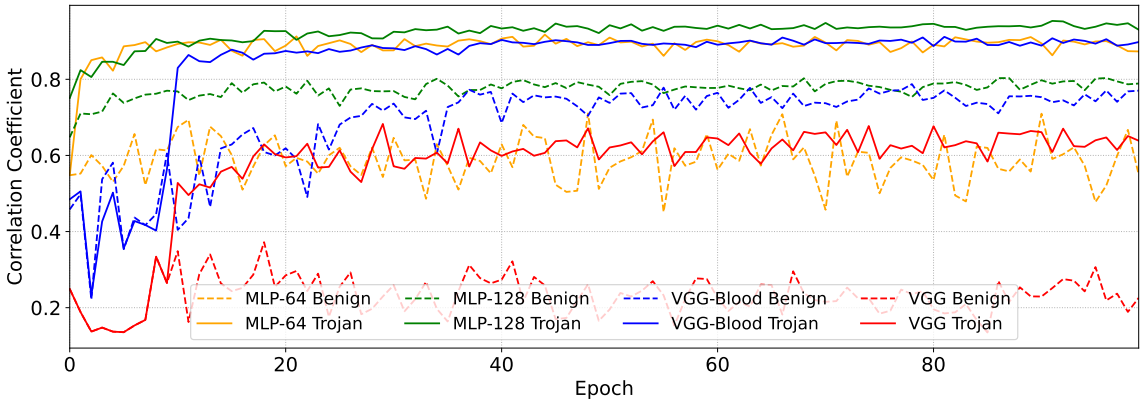


Figure 4.5.: Correlation coefficient of power estimate from MAC operation and classification results over 100 training epochs.

4.1.3.1. Output Recovery on Estimated Power

For the power estimates derived from the power model during GPU-based inferences, a simple random forest classifier was sufficient to achieve high accuracy. The parameters for the random forest were optimized using a grid search. With emulated power consumption data, we achieved up to 99% accuracy in recovering output labels when the Trojan was implemented on an MLP with 256 neurons. Even for a smaller network with only 64 neurons, the recovery accuracy remained robust at 91%.

In comparison, the random forest achieved 84% accuracy when classifying output labels from the benign network's estimated power consumption. For the VGG models, the insertion of the Trojan proved more challenging with the CIFAR-10 dataset, likely due to the benign model only achieving a 82% test accuracy. Nevertheless, the Trojan significantly boosted both the correlation and output recovery, as detailed in Table 4.2.

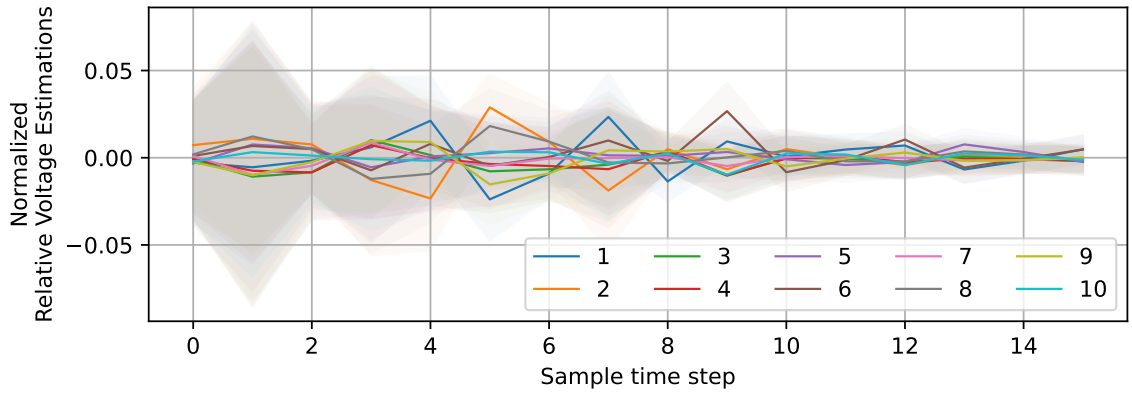
Next, we evaluated whether this enhanced recovery performance extended to actual hardware implementations, providing further insights into the Trojan's impact.

4.1.3.2. Output Recovery on Power Traces

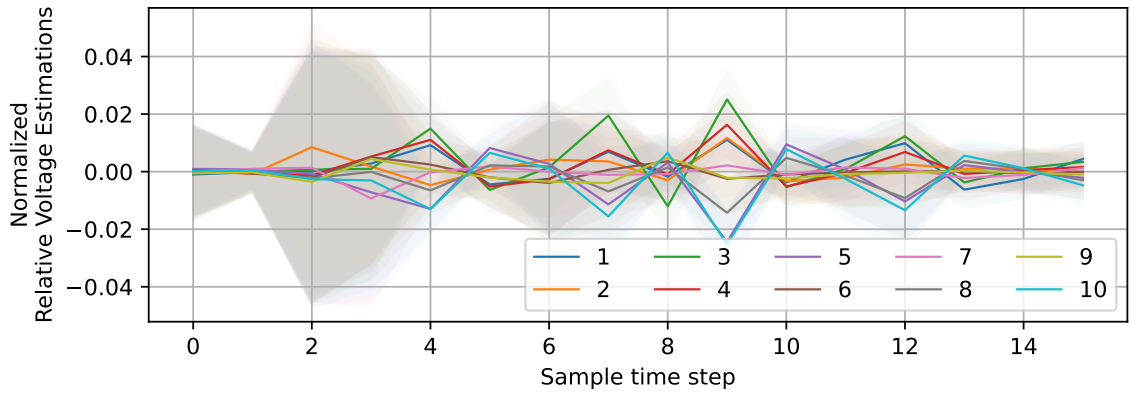
For the first experiment on hardware, we analyzed the power traces from the benign MLP-64. With 10× averaged power traces from this accelerator, our classifier achieved around 67% accuracy. Additionally, power traces from the benign models were compared with the Trojan models for the MLP-64 and VGG (BloodMNIST).

4.1.3.3. Comparison of Power Trace Variability Between Benign and Trojan Models

We show the average voltage measurements of 10 k traces, while each of the 1–10, or 1–8 for BloodMNIST, different output class labels are computed in Figures 4.6 and 4.7, respectively. The traces are normalized to a range of [0,1]. The highlighted area around the traces illustrates the overlap of the standard deviation from traces of different classes. It is apparent that the benign MLP-64 (Figure 4.6a) exhibited greater variations in measurements in comparison to the Trojan MLP-64 (Figure 4.6b), which narrowed down due to the training process. We also note that the average traces split up much more for the Trojan model, which became most noticeable at time step 9 of the MLP-64. The same behavior can be observed for the Trojan VGG (BloodMNIST) in Figure 4.7, for which, in some areas, the standard deviation was only half of that of the benign network. This can be better observed in Figure 4.7c,d, which show the measurements for the first 1024 sample time steps for class 5. Here, it is also visible that the power profile of the Trojan model was much more pronounced with larger fluctuations than that of the benign model, which made it more robust against noise. The classifier could learn traces from the Trojan model more easily, as the traces were closer to the mean of their corresponding output label and more different from each other. Otherwise, the traces were seemingly similar, making it impossible to distinguish a Trojan network from a benign one through simple power analysis.



(a)



(b)

Figure 4.6.: Excerpt of normalized power traces from the inference of a benign and Trojan MLP-64, averaged by the accelerator output. Different colored lines denote different class labels, the standard deviation for each class is highlighted in the respective colors. (**a**) Benign MLP-64; (**b**) Trojan MLP-64.

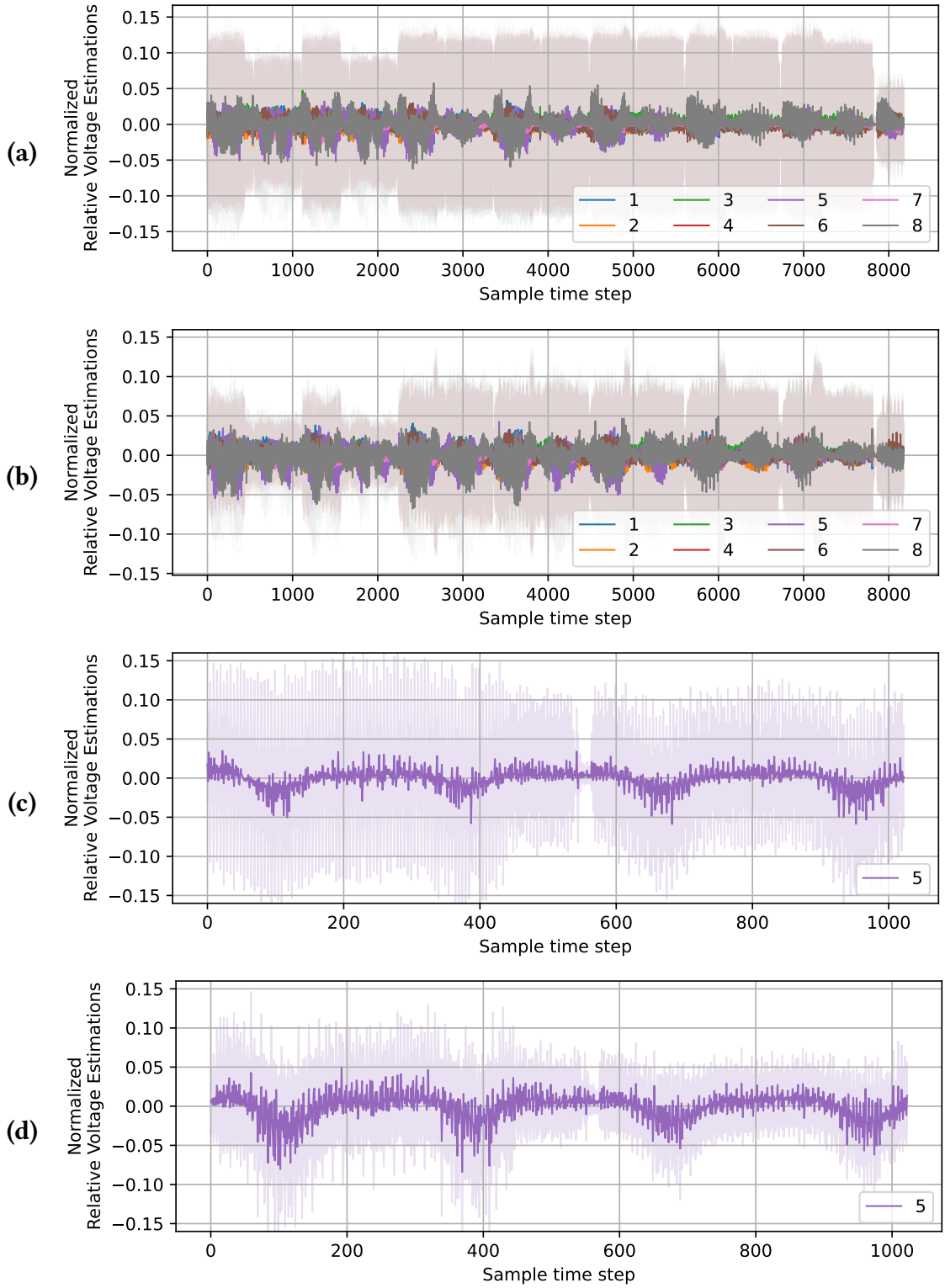


Figure 4.7.: Excerpt of normalized power traces from the inference of a benign and Trojan VGG (BloodMNIST), averaged by the accelerator output. Different colored lines denote different class labels, the standard deviation for each class is highlighted in the respective colors. (**a**) Benign VGG (BloodMNIST); (**b**) Trojan VGG (BloodMNIST); (**c**) Benign VGG (BloodMNIST) zoomed in to the first 1024 samples for class 5; (**d**) Trojan VGG (BloodMNIST) zoomed in to the first 1024 samples for class 5.

Table 4.2.: Summary of results with 10× averaging, showing the accuracy score for the output recovery attack. "-" indicates same value as previous row.

Dataset	Network	Temp.	MAC-Output Correlation		Output Rec. (Power Model)		Output Rec. (Hardware)	
			Benign	Trojan	Benign	Trojan	Benign	Trojan
MNIST	MLP-64	Room	0.55	0.87	84%	91%	67%	81%
		0 °C	-	-	-	-	65%	74%
		40 °C	-	-	-	-	67%	72%
	MLP-128	Room	0.78	0.93	93%	98%	72%	87%
		0 °C	-	-	-	-	70%	86%
		40 °C	-	-	-	-	68%	83%
	MLP-256	Room	0.46	0.96	94%	99%	56%	89%
		0 °C	-	-	-	-	59%	84%
		40 °C	-	-	-	-	55%	78%

Table 4.2.: *Cont.*

Dataset	Network	Temp.	MAC-Output Correlation		Output Rec. (Power Model)		Output Rec. (Hardware)	
			Benign	Trojan	Benign	Trojan	Benign	Trojan
Speech Commands	KWS-256	Room	0.36	0.64	83%	89%	71%	80%
		0 °C	-	-	-	-	70%	80%
		40 °C	-	-	-	-	69%	79%
CIFAR-10	VGG	Room	0.22	0.63	26%	56%	42%	75%
		0 °C	-	-	-	-	43%	71%
		40 °C	-	-	-	-	40%	69%
Blood MNIST	VGG	Room	0.77	0.89	74%	81%	73%	87%
		0 °C	-	-	-	-	71%	85%
		40 °C	-	-	-	-	67%	83%

Table 4.2 presents the final accuracy scores for the classifier trained on data with 10× averaging. The classifiers were evaluated separately for each temperature, using the same split of training and test data, ensuring that each test sample corresponded to three distinct power traces. By incorporating power traces with varying temperatures during training, we enhanced the classifier's robustness against such variations. This approach proved particularly effective for the KWS-256 model, which exhibited no significant drop in accuracy, even when measurements were conducted at 40 °C or 0 °C.

4.1.3.4. Impact of Model Size and Configuration on Trojan Effectiveness

The best results were observed with the MLP-256 model using a 4 × neuron fold configuration, where 64 MAC operations were executed in parallel. Compared to the MLP-64 model, which produced four distinct points of interest in the power trace (each corresponding to 64 MAC operations), the MLP-256 generated 16 points of interest, while maintaining the same number of MAC operations. This increase in the number of points of interest provided more detailed power trace data, enhancing the Trojan's effectiveness. For models trained on the MNIST dataset, we observed that the Trojan's impact became more pronounced as the number of neurons per layer increased (64, 128, 256). Larger models, like the MLP-256, exhibited higher output recovery success rates, due to the richer parameter space and more distinguishable power traces. However, this trend did not hold for the benign models. For instance, the output recovery success rate for the benign MLP-256 was lower than that of the benign MLP-128 and MLP-64, indicating that the Trojan specifically exploited the structural advantages of larger models, while the benign models remained less susceptible to such attacks.

To show that our method worked, we wanted to achieve an accuracy of at least 75%, while using the least amount of averaging required for this goal. Thus, we evaluated how the attack scaled for different amounts of averaging. The results are presented in Figure 4.8 with 1× (no averaging) to 10× averaging. We can notice a consistent increase in accuracy with higher numbers of averaging for all of the Trojan models and that the recovery scaled better for larger models. For the MLP-128, MLP-256, KWS-256, and for the VGG (BloodMNIST), we needed only 2× averaging and 10× averaging for the remaining models to achieve an accuracy of at least 75%.

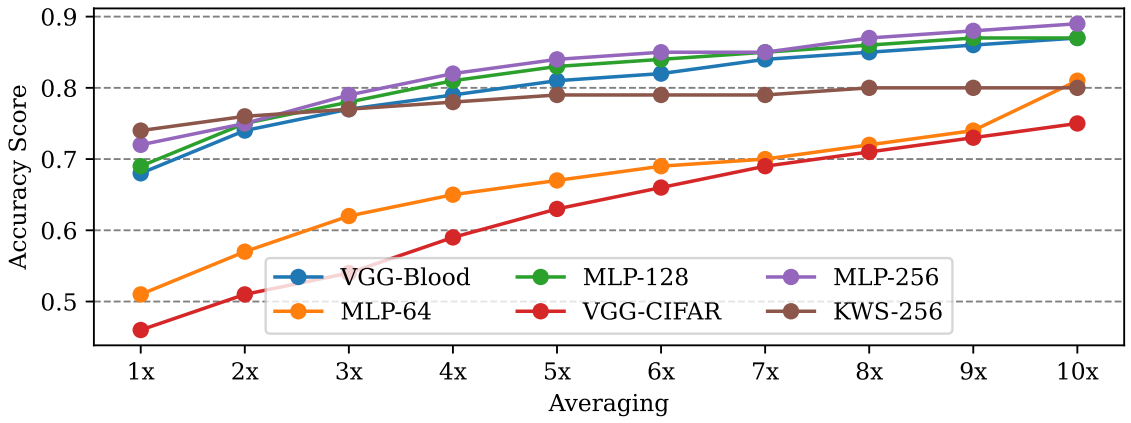


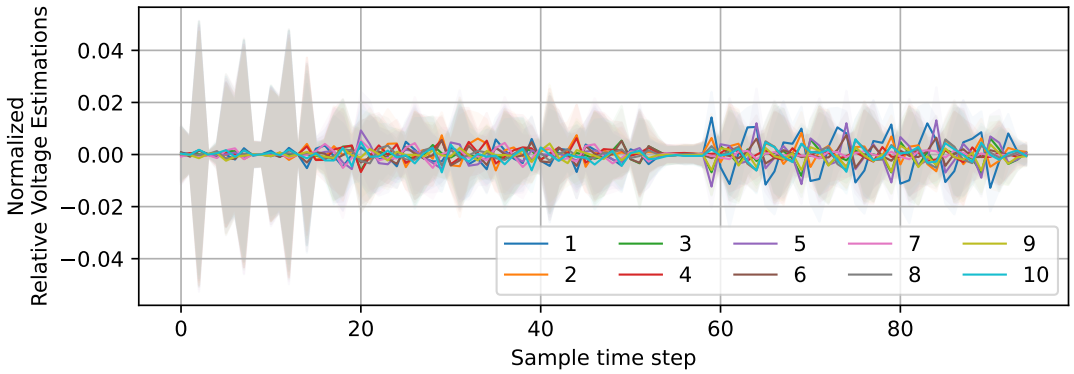
Figure 4.8.: Comparison of the effect of different amounts of averaging on the accuracy of the output recovery with measurements taken at room temperature.

4.1.3.5. Extended Experiments on MLP-256

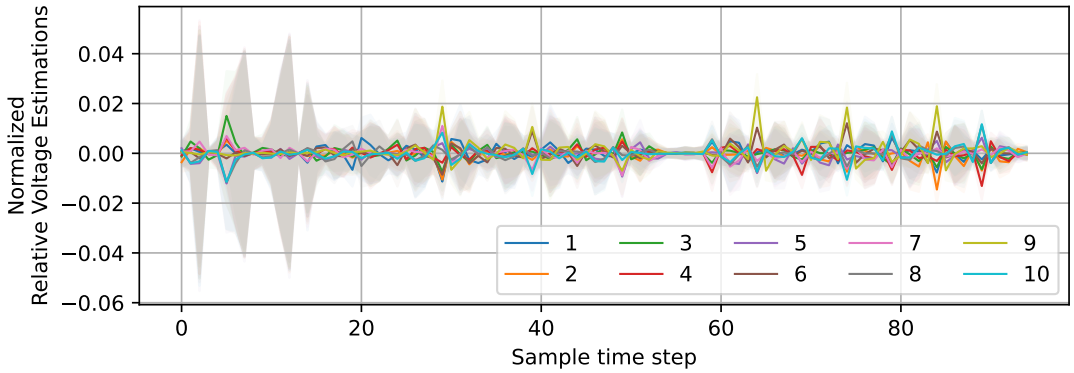
Building upon our previous experiments, we further investigated the quality of power traces in relation to the output recovery, focusing on the impact of significantly increased

averaging. For this analysis, we selected the MLP-256 model trained on the MNIST dataset, as it demonstrated the highest improvement in output recovery accuracy, while incurring only a minimal 2% reduction in test accuracy. As detailed in Table 4.2, the application of the Trojan increased the hardware output recovery rate from 56% to 89%, with just 10× averaging. To explore the limits of this improvement, we extended our analysis by employing up to 100× averaging on both the benign and Trojan-affected models, to evaluate their respective performance and the potential for further enhancements in recovery accuracy.

We present the resulting power traces averaged by the corresponding output class for the benign and Trojan model in Figure 4.8. Similarly to our findings in Figure 4.7, it can be seen that the standard deviation was reduced for the Trojan model and that the traces split up at several time steps during the inference. The figure also shows that our hardware-aware training increased the distance between the measurable voltage fluctuations of the different classes. While for the benign model (a) only the average trace for class 1 stood out from the remaining classes, there were still remarkable overlaps. In the case of the Trojan (b), when the traces split up, the amount of overlaps was low.



(a)



(**b**)

Figure 4.8.: Excerpt of normalized power traces from the inference of a benign and Trojan MLP-256 with 100× averaging, grouped by accelerator output. Different colored lines denote different class labels, the standard deviation for each class is highlighted in their respective colors. (**a**) Benign MLP-256; (**b**) Trojan MLP-256.

Additionally, Figures 4.9 and 4.10 show the average inter- and intra-class absolute distance of the measurements at 10× averaging and 100× for the benign and Trojan model. As a measure of dissimilarity, the inter-class distance d_{inter} was calculated as the point-wise absolute distance between the average of traces $\overline{\mathbf{x}^c}$ of each class c. C is the number of classes.

$$d_{inter} = \frac{1}{C}\sum_{c}^{C}\frac{\sum_{c'\neq c}^{C}|\overline{\mathbf{x}^c} - \overline{\mathbf{x}^{c'}}|}{C-1} \tag{4.7}$$

On the other hand, as a measure of similarity, the intra-class distance d_{intra} was calculated as the mean of point-wise absolute distances of traces $\mathbf{x}^c$ of the same class c. $\mathbf{x}_i^c$ is trace i of class c, and N_x is the number of combinations of traces without duplication.

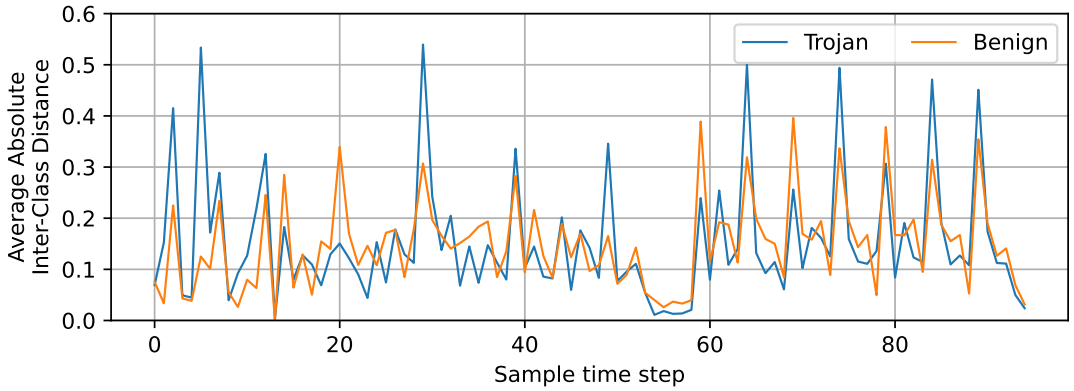
$$d_{intra} = \frac{1}{C}\sum_{c}^{C}\frac{\sum_{i<j}|\mathbf{x}_i^c - \mathbf{x}_j^c|}{N_x} \tag{4.8}$$

Trojan model is expected to have a lower point-wise intra-class distance than the benign models, as the power consumption for traces of the same class should be similar.

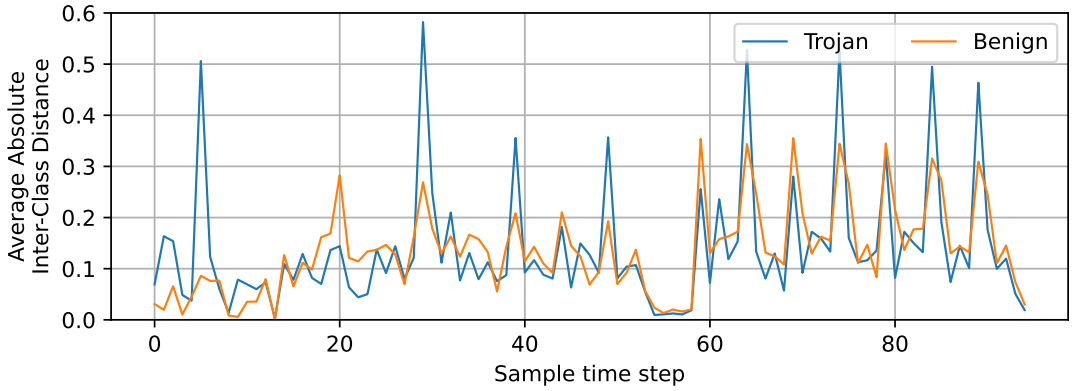
For the intra-class distance, the Trojan model had lower values in the majority of time steps, and the values also decreased more drastically with more averaging in comparison to the benign model. At the same time, we noticed that the inter-class distance for the Trojan increased in many time steps, while it remained the same or decreased for the benign model. The spikes seen in Figure 4.9 correspond to the samples in Figure 4.11, for which it is easiest to distinguish between classes.

Finally, experiments with averaging increased to up to 100× were performed, and the results are presented in Figure 4.11. Here, we compared the accuracy of the classifier trained on power traces from the benign and Trojan-injected MLP-256 models across varying

levels of averaging. As seen by the blue lines for the Trojan model and orange lines for the benign model, the Trojan benefited more from better measurements, especially for 0 °C and 40 °C, for which we gained 9% and 15%, respectively. For the Trojan model measured at room temperature, we gained an additional 7% for the final accuracy, ending up at 96%. The accuracy for the benign model only increased up to 62% for room temperature and 0°C and 63% for 40 °C.

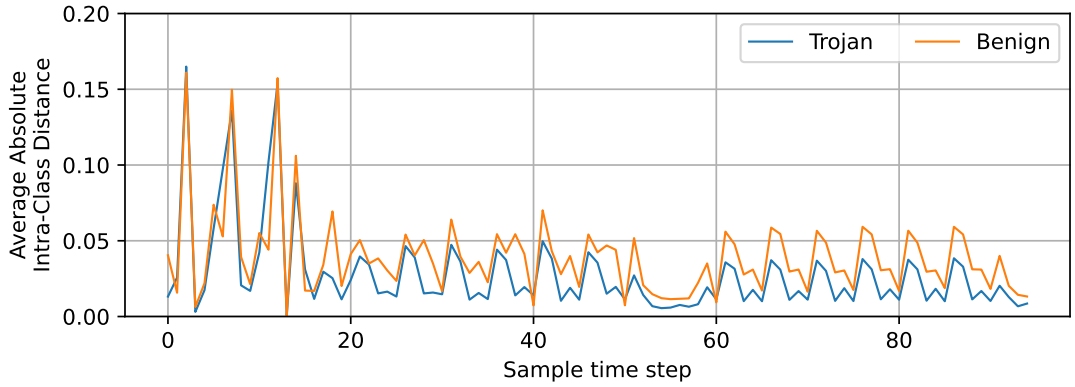


(**a**)

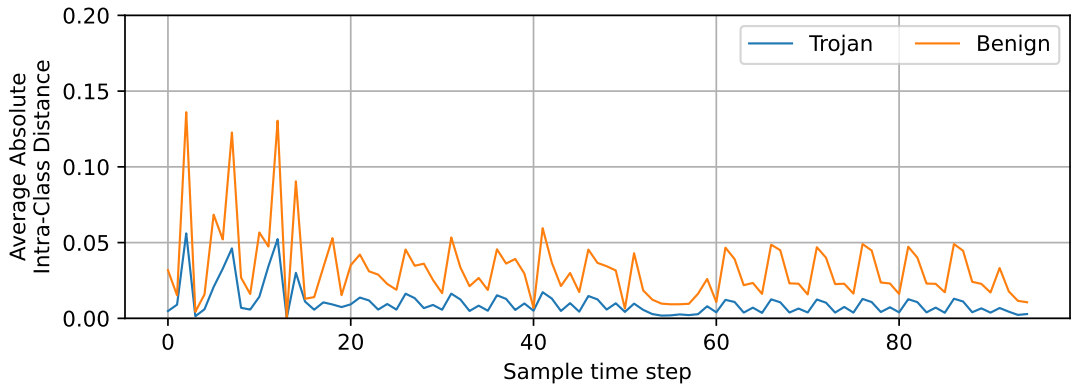


(**b**)

Figure 4.9.: Per sample time step inter-class average absolute distance of the measured power traces grouped by the type of model for the MLP-256. Inter-class was calculated as the distance between classes. (**a**) Inter-class average absolute distance at 10× averaging; (**b**) inter-class average absolute distance at 100× averaging.



(a)



(b)

Figure 4.10.: Per sample time step intra-class average absolute distance of the measured power traces grouped by the type of model for the MLP-256. Intra-class was calculated from the samples of a class. (**a**) Intra-class average absolute distance at 10× averaging; (**b**) intra-class average absolute distance at 100× averaging.

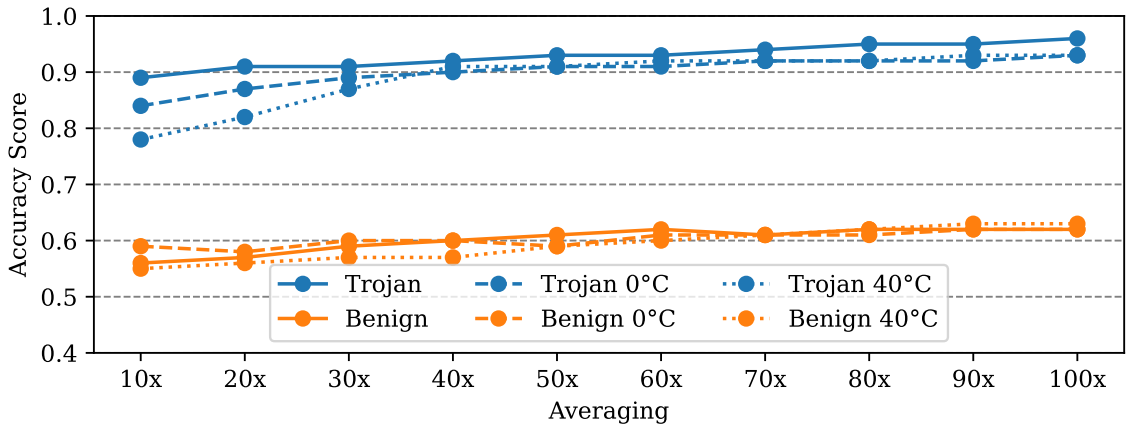


Figure 4.11.: Comparison of the effect of 10–100× averaging on the accuracy of the output recovery with measurements taken at different temperatures.

4.1.3.6. Robustness of the Trojan to Fine-tuning

Beyond demonstrating the effectiveness of the Trojan attack, we explored the robustness of the malicious models against attempts at removal and retraining. In this scenario, we assumed that the victim had a detailed understanding of the training process as described in this paper, enabling them to undertake efforts to counteract the Trojan's effects. To further narrow the hyperparameter search space for the victim, we provided them with knowledge of the specific values chosen for the Trojan injection parameters: α_{push}, α_{pull}, and α_{corr}. This assumption represents a worst-case scenario for the adversary, where the victim is fully informed about the injection strategy.

Additionally, the FINN's functionality for reading and modifying network weights at runtime provided a theoretical pathway for the victim to extract the model weights directly from the hardware and fine-tune the network. This fine-tuning process could potentially mitigate the malicious effects of the Trojan by altering the trained parameters. However, it is important to note that the ability to extract and adjust weights depends on whether the adversary has enabled this option during the deployment phase. If such access is restricted, the victim's ability to counteract the Trojan becomes significantly limited.

For fine-tuning, we considered continued training with the data from the training set for another 100 epochs. For the model, we chose the MLP-256 trained on MNIST. Then, we adjusted the hyperparameters to remove the Trojan, for which we considered the following three scenarios:

1. Training with $\alpha_{\text{push}} = 0$ and $\alpha_{\text{pull}} = 0$ (neglecting the switching activity)
2. Training with reversed α_{push} and $\alpha_{\text{pull}} = 0$
3. Training with reversed α_{pull} and $\alpha_{\text{push}} = 0$

Our findings indicated that attempting to reverse both α_{push} and α_{pull} simultaneously led to an unacceptable degradation in the model's classification accuracy. As a result, our experiments focused on scenarios where one of these parameters was set to 0, allowing us to evaluate the impact of adjusting each parameter individually.

In **scenario 2**, rather than pushing the switching activities of different output classes further apart, the goal was to pull them closer together. This adjustment aimed to reduce the distinguishability of the power traces across the different classes, thereby counteracting the malicious training effect and making it harder for an attacker to extract useful information from the side-channel leakage.

For **scenario 3**, the focus shifted to increasing the intra-class distance, which corresponded to a greater variability within the power traces for a single class. By doing so, the standard deviation for each class was intentionally increased, introducing noise into the patterns used by the attacker. This higher variability within classes not only complicated the attack but also disrupted the underlying correlation between the power traces and the classification outputs, significantly reducing the effectiveness of the Trojan.

Initially, we observed that the model accuracy deteriorated when setting the hyperparameters to a higher value than in the malicious training. A value of $1 \cdot 10^{-4}$ for α_{push} or α_{pull} resulted in an accuracy drop of 3%. Table 4.3 shows the results of the side-channel based output recovery for the fine-tuned Trojan MLP256. Manipulating the model based on α_{push} had the strongest effect on the Trojan, with a drop of 6%; however, the attack accuracy was still 27% higher than on the benign model. For the most realistic case, which was plain training without considering the Trojan training, the output recovery barely degraded, while the accuracy of the model slightly increased.

From these experiments, we concluded that fine-tuning, even when the victim was given the advantage of access to training data and knowledge about the malicious training method, was not sufficient for removing the Trojan from the model. To not degrade the accuracy of the model, the victim was limited to subtle modifications to the model, which did not decrease the potency of the side-channel-based output recovery.

Table 4.3.: Model accuracy and output recovery in hardware for the fine-tuned Trojan MLP256. Parameters were chosen to avoid accuracy degradation for a baseline of 95% classification accuracy.

α_{push}	α_{pull}	Model Accuracy	Output Recovery
0	0	96% (+1%)	87% (−2%)
$-1 \cdot 10^{-5}$	0	95%	83% (−6%)
0	$-1 \cdot 10^{-5}$	95%	85% (−4%)

4.1.4. Section Discussion

In this work, we explored the vulnerability of neural network FPGA accelerators to classification result leakage via power side-channel attacks. By default, extracting the output of a neural network through power analysis is inherently challenging, due to the weak correlation between the power consumption and the network's output labels. However, we demonstrated that this limitation can be overcome by deliberately manipulating the training process of neural networks to amplify this correlation. Our approach introduces a hardware-aware training methodology that incorporates additional loss functions specifically designed to account for a power model of the neural networks MAC operations. These loss functions are engineered to optimize the network parameters in a manner that significantly enhances the correlation between the power consumption patterns and the network's output labels during inference. As a result, the power consumption of such a Trojan-injected model leaks considerably more information about its classification results when executed on FPGA-based accelerators. While evolutionary modeling was not explored in this work, it holds significant potential for effectively injecting Trojans into benign models post-training. This approach will be considered as a direction for future research.

Trojan-enhanced neural networks function identically to benign networks in terms of accuracy and behavior, and they do not require additional hardware resources. This stealthy

modification ensures that the malicious alterations remain undetectable during standard operations. Furthermore, the differences in power traces between benign and Trojan models are so subtle that identifying them without direct side-by-side comparisons is nearly impossible. Our findings underscore the critical risks posed by Trojan side-channels that can be covertly embedded within a network's parameters. These vulnerabilities enable adversaries to compromise user privacy, without impacting the observable performance or behavior of the system. In conclusion, our work highlights the urgent need for developing robust countermeasures to detect and mitigate Trojan side-channels, safeguarding neural network accelerators from such covert threats.

4.2. EvoWeight: Sponge Poisoning of FPGA-Based DNN Accelerators in Differential Private Secure Federated Learning

The work described in this chapter was published in [10] and is joint work with co-authors Muhammad Shakeel Akram, Mehdi Tahoori, Bogaraju Sharatchandra Varma and Dewar Finlay.

Continuous retraining of Machine Learning (ML) models is essential to maintain accuracy and generalization. Recent approaches employ Differential Private Secure Federated Learning (DP-Secure FL), enabling collaborative training through privacy-preserving updates without sharing raw data [131], [132]. This paradigm is particularly relevant for privacy-sensitive domains such as healthcare and addresses challenges in reliability, security, and efficiency [133], [134], [135].

The integration of FPGAs into DP-Secure FL further enables deployment in resource-constrained edge environments, offering benefits such as low power consumption, high throughput, and efficient hardware-software co-design [136], [137]. However, the privacy-preserving nature of DP-Secure FL also complicates the detection of internal adversaries, such as free-riders injecting malicious updates [138], [139].

In this context, sponge poisoning attacks [140], [141], [142] aim to increase power consumption without degrading model accuracy. On FPGA-based edge devices, such attacks are particularly harmful due to limited monitoring capabilities and tight power budgets. Increased power consumption leads to higher temperatures, reduced device lifetime, and degraded system performance, potentially causing denial-of-service conditions.

This work introduces a novel sponge poisoning attack targeting FPGA-based DP-Secure FL systems. Instead of manipulating inputs, the attack crafts model weights to increase neuron activity and power consumption while preserving accuracy. This enables adversaries to exploit the system without active participation during training and bypass existing countermeasures.

The main contributions are as follows:

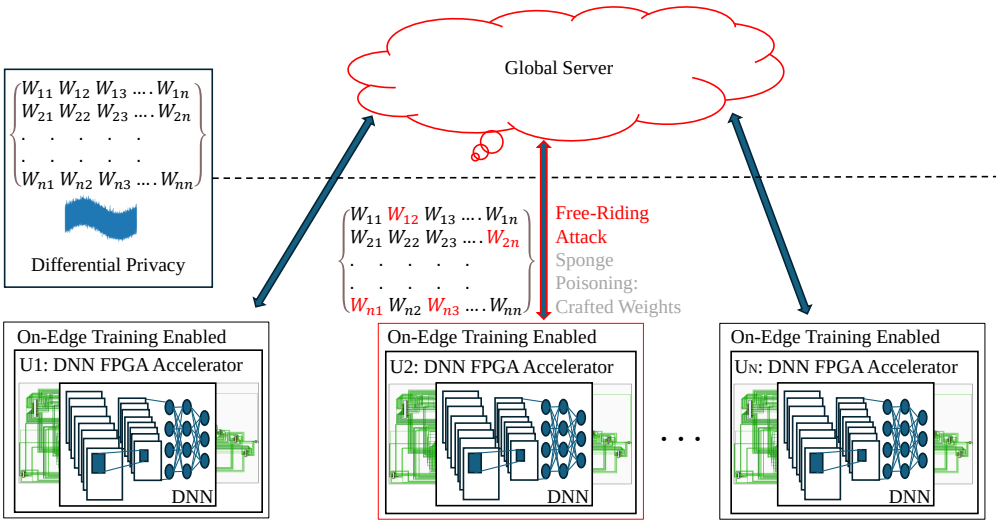


Figure 4.12.: N users in a DP-Secure FL setting with Possible Free Rider Attack. U2 is an adversary that does not participate in the training of the model but instead pushes poisoned or random weights.

- First sponge poisoning attack targeting power consumption in FPGA-based DP-Secure FL through malicious weight sharing
- First demonstration of sponge poisoning on binary DNNs, revealing vulnerabilities despite their efficiency
- Novel weight-crafting techniques that increase power consumption while maintaining model accuracy

The remainder of this section is taken directly from [10] and is reproduced verbatim.

4.2.1. Adversarial Activities in DP-Secure FL

Despite its robustness, DP-Secure FL complicates the identification of malicious users, whether these are users or the global server, and challenges for effectively detecting internal threats. In the following, the security threats arising from the anonymity of DP-Secure FL are summarized.

Untrusted Global Server: While an untrusted server in FL is generally unlikely, it could initiate an attack to manipulate the models of specific users. If the attack is random and affects a limited number of users, it becomes nearly undetectable within the DP-Secure FL setup. Nonetheless, the server remains a single point of failure in the system.

Malicious User and Free-Rider Attacks: Malicious users, especially those acting as free-riders pose a greater threat. Free-rider as illustrated in Figure 4.12, contributes minimal or random updates to the server but benefits from aggregated model improvements. Literature suggests that while DP-Secure Aggregation effectively prevents most external attacks, it is vulnerable to free-rider attacks due to their nature to ensure the privacy of the users and

data [143], [144], [145], [146]. This enables the adversary to design and embed various legitimate and even benign-looking constructs in their designs to perform free-rider attacks, evading many detection mechanisms [139].

Free-rider attacks bring several adverse effects in a DP-Secure FL setting:

- *Degraded Model Quality*: Free-riders submitting random updates may contribute to a decline in model accuracy if their proportion in FL is significant.
- *Reduced Model Generalization*: With contributions from non-diverse, random data, the aggregated model becomes less generalizable.
- *Wasted Resources*: Free-riders waste communication and computational resources, increasing power usage and memory consumption on the server in the FL setting.
- *Increased Vulnerability*: Free-riders open the system to other attacks, such as injected watermarks, sponge poisoning, etc., remaining invisible over time.

4.2.2. Sponge Poisoning in Neural Networks

Sponge poisoning in neural networks is a type of adversarial attack in which a maliciously crafted model (or inputs) is designed to be highly inefficient during inference, effectively "sucking up" computational resources like a sponge [140]. This attack benefits from the inefficiencies in specific layers or operations of the model, causing unusually long processing times and excessive power consumptions, without degrading model accuracy to remain undetected and not mitigated by stateful defences. Sponge poisoning typically involves generating adversarial perturbations across multiple test samples, making it computationally expensive. An attacker continuously generates new sponge examples until the system is significantly slowed down [140], [142], [147], [148].

Existing sponge poisoning methods are limited to specific inputs and do not consider manipulating the weights of the model to achieve higher power consumption. Thus, they are not suitable for a DP-Secure FL setting, where the data is not shared among users. Additionally, they lack evaluation on quantized neural networks, such as B-DNNs, which are usually utilized on low-power edge devices, where increased power consumption would be most critical.

4.2.3. Free-Rider Sponge Poisoning in DP-Secure FL

In this work, we develop a free-riding attack, as illustrated in Figure 4.13, consisting of the following steps:

1. *Initial collaboration:* All users actively participate in collaborative learning within a DP-Secure FL.

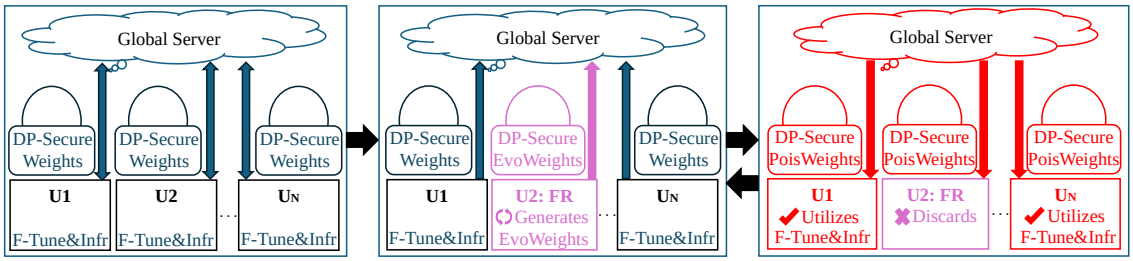


Figure 4.13.: EvoWeight Sponge Poisoning in FPGA-based DP-Secure FL. Initially, all participants take part in the training until a certain desirable accuracy is reached. Then, the adversaries (U2) generate malicious EvoWeights which are spread to all other participants through the aggregation in the global server. The adversaries discard the poisoned weights and keep the latest clean version.

2. *Free-Rider's deviation from honest collaboration:* After receiving the updated weights from the global server, the free-rider generates one-time malicious EvoWeights and submits them as user updates to the server.
3. *Aggregation and distribution:* The global server aggregates the weights and distributes the resulting poisoned weights (PoisWeights) back to all users.
4. *Impact on honest users:* The free-rider discards the PoisWeights, while honest participants experience increased power consumption and degraded performance metrics, such as throughput, runtime, and inference time.

This cycle repeats as the free-rider continues submitting pre-generated EvoWeights in subsequent steps, avoiding the overhead of crafting new weights while maintaining the attack's impact. This enables sponge poisoning for DNNs in a DP-Secure FL setting, demonstrating the imminent danger from adversaries to embed seemingly legitimate constructs in their updates, thereby executing free-riding attacks to exploit the FL process while evading many detection mechanisms, ultimately impacting the system's reliability, efficiency, and device lifetime in the FL environment.

4.2.3.1. Weight crafting over input poisoning

Instead of creating new sponge examples for every instance, we manipulate shared weights directly in a DP-Secure FL setting, increasing the number of firing neurons and amplifying energy consumption and latency during the fine-tuning process on the FPGA's Processing System (PS) and inference on the DNN accelerator within the Programming Logic (PL). This hardware-software co-design approach, integral to DP-Secure FL, may have its benefits undermined by this attack, leading to faster battery drainage, increased prediction latency, reduced throughput, and compromised availability to legitimate users [147], [149]. The adversary can even tune the attack to avoid exceeding specific energy consumption limits, helping the free-rider remain undetected [148]. This sustained high energy usage can elevate hardware temperatures, triggering throttling mechanisms in modern systems that further degrade performance and increase latency. Compared to related work, which relies on crafting sponge examples, our attack is effortless for the adversary, as costly backpropagation-based optimization is not required and no training samples have to be

acquired. Furthermore, weight-dependent sponge poisoning is always active and like a virus spreads to all targets through updates in the FL.

4.2.3.2. Overcoming hardware as a black-box

In DP-secure FL, the global model architecture is pre-determined for the users, allowing the free-rider to exploit according to the folding configuration of users. Since folding factors are typically divisors of the number of neurons, a free-rider can optimize sponge poisoning by aligning crafted gradients for maximum possible parallelism in any participant's FPGA implementation. When the FPGA-based DP-Secure FL setup is built on FINN [31], the optimization challenge is further reduced, given the predefined list of supported FPGA boards such as Pynq-Z1, Pynq-Z2, Kria SOM, Ultra96, ZCU102, ZCU104, and Alveo cards. Knowing the specific hardware configurations and supported boards enhances the attacker's ability to tailor the sponge poisoning strategies, potentially enhancing the power consumption effects without degrading model accuracy. Manipulating the model weights for crafting sponge attacks can also bypass ASIC optimizations, as attackers target weight modifications to enhance the number of firing neurons, rendering traditional FL security protocols ineffective.

4.2.3.3. Bypassing security protocols

We show that existing security protocols in FL often fall short against free-rider sponge-poisoned attacks with well-aligned crafted weights. DP-secure aggregation involves the global server receiving aggregated weight updates from local devices, anonymized by Gaussian noise addition to prevent deducing specific model updates and safeguarding participant privacy and security. This hinders the detection of free-rider attacks. Fine-tuning is often employed to mitigate the effects of poisoned updates by iteratively refining the global model, at the cost of increased computational resources and representative training data [150]. Our findings demonstrate that well-aligned EvoWeights can bypass or diminish the effectiveness of fine-tuning, highlighting a critical vulnerability in such settings.

Detecting free-riders via outlier analysis, which identifies updates deviating from the norm or having a low variance, is a common mitigation strategy. However, this approach becomes challenging when weight crafting does not affect model accuracy. Similarly, sanitization methods for countering such attacks may prove too costly for practical deployment [148].

4.2.4. Techniques for EvoWeight

In the context of FL, where only model weights are exchanged between users and a global server, sixteen targeted experiments were conducted to explore weight crafting techniques. The first experiment (E0) involved manually crafting a small subset of weights to test power consumption. The second experiment (E1) employed automated weight crafting to

maximize neuron activation incrementally, progressing from 0% to 100% inactive neurons in a random sequence. The remaining fourteen experiments (E2-E15) employed automated weight crafting on top of the E0 setup to maximize neuron activation incrementally in various sequences. Methods included random activation (E2), ascending and descending patterns, diagonal and alternate activations, algorithmic patterns from the center and edges, quadrant-wise patterns, and clustered activations with varying cluster sizes, such as 5, 10, 15, 20, and 30. The final experiment (E15) combined all techniques' highest achieved firing neuron percentage.

The experiments aimed to activate previously inactive neurons, thereby inducing power-increasing effects (sponge poisoning) without compromising model accuracy. Instead of crafting inputs to influence weights indirectly, each experiment directly altered the weights to identify and selectively activate neurons—referred to as "firing neurons (FNs)." The pre-built accelerator was directly updated with crafted weights in the hardware setting, as detailed in our previous work[151]. Each experiment involved assessing the top-1 test accuracy across 45270 samples to identify the highest percentage of FNs in each layer of DNNs without affecting accuracy.

Evolutionary Algorithm for EvoWeight: The experiments utilize the evolutionary algorithm [152], as illustrated in Algorithm 1. Starting with an initial population of crafted weights (0% to 100%) for each layer, the fitness function evaluates their performance while maintaining a threshold accuracy to prevent sponge poisoning from detection. Subsequent generations are produced by selecting the layer-wise best-performing solutions, by applying genetic operators—specifically mutation and crossover. The mutation operator introduces variability by modifying the weights, thereby exploring the solution space. The crossover operator combines layer-wise solutions to generate the most effective crafted weights, thereby uses the strengths of multiple solutions to create potentially superior solutions. This activation process was carefully managed to ensure that accuracy remained within acceptable bounds, as any participant weight updates causing a significant accuracy drop are either detected or rejected in FL setups. This lightweight nature of the evolutionary algorithm enables the free-rider to craft weights concurrently with model training on edge devices within the DP-Secure FL environment. Once crafted, these optimized weights can be seamlessly deployed in subsequent global rounds of the DP-Secure FL, introducing power attacks for honest users.

Each experiment measured power consumption across 45,270 samples for top-1 accuracy to assess the effectiveness of sponge poisoning. The findings reveal that this method can be used not only to measure power but also to potentially improve model accuracy. Certain crafting techniques yielded higher accuracies, presenting an efficient real-time training alternative for FPGA accelerators. By tuning weight configurations, pre-trained accelerators can be enhanced without requiring resource-intensive backpropagation, providing a streamlined approach to model improvement.

Algorithm 1 EvoWeight Algorithm

Input: Clean weights $\mathcal{P}$, threshold accuracy τ, DNN layers L, mutation rate r_{mut} covering a range of weights (0% to 100%).
Output: Optimized crafted weights $\mathcal{W}^*$.
Initialize: Utilize $\mathcal{P} = \{W_1, W_2, \ldots, W_n\}$
for $i = 1$ to L **do**
    Mutation: $\Delta\mathcal{P}$ at r_{mut} for increased firing neurons.
    Evaluate: Fitness for each solution $i_{W'}$ based on the
    power consumption and accuracy.
        if Accuracy $< \tau$; Discard solution $i_{W'}$.
    Selection: Select the top k solution based on fitness.
    Crossover: Combine layer-wise selected solutions.
    Update Population: Replace $\mathcal{P}$ with top-performing
    offspring solution.
end for
Evaluate Effectiveness: Measure power consumption across 45,270 samples for top-1 accuracy to assess the effectiveness of sponge poisoning.
Return: Best-performing crafted weights $\mathcal{W}^*$.

4.2.5. Experimental Setup

The proposed design is implemented and evaluated on the Xilinx AUP PYNQ-Z2 [153]. The design workflow uses tools such as Xilinx Vivado and FINN [31] for accelerator development and PyTorch [33] and Brevitas [34] for deep neural network (DNN) design. For arrhythmia classification, the Physionet MIT-BIH Arrhythmia Dataset ECG [154], [155] is used, identifying five types of cardiac beats—normal, supraventricular premature, ventricular escape, fusion of ventricular and normal, and unclassifiable beats—as part of efforts to address cardiac disease (CD) risks affecting over 50 million people globally [156]. The dataset contains 339,390 samples, partitioned with the global server holding 50,000, 8,824, and 9,054 samples for training, validation, and testing, respectively, while the remaining data is distributed among individual users.

The classification task is performed in a DP-secure FL environment using a Binarized DNN Accelerator (B-DNN-Accel), exploring weight-crafting techniques, and sponge poisoning attacks. The accelerator build on B-DNN includes each hidden layer, along with their activation layers, and weights set to 1-bit quantization, achieving a top-1 test accuracy of 81.65%.

To analyze the sponge poisoning on FPGA accelerators accurately, a shunt resistor is used for precise power measurements, as smaller boards like the PYNQ-Z2 lack PMBus [157] support. In typical low-load scenarios, idle current can be high due to IO pins not being pulled low or low-power states not being activated. At such low logic element (LE) usage, power measurements are challenging to capture accurately, particularly in the milliampere range, which excludes additional switching behaviour. Therefore, practical FPGA power

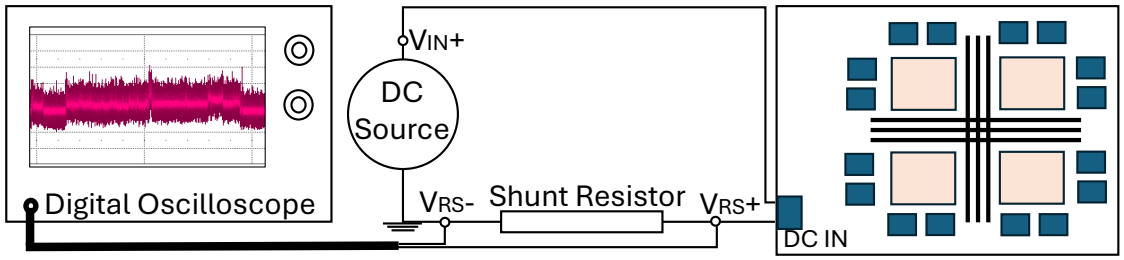


Figure 4.14.: Power Measurement Setup

measurements often incorporate hall-effect sensors at DC or PCIe insertion points, high-side shunt sensing, or embedded current-sensing DC-DC converters with I2C interfaces for improved accuracy.

A shunt resistor approach provides a direct measurement of power consumed by the board, which is essential for capturing real-time current fluctuations during attacks. Figure 4.14 illustrates the measurement setup, where a shunt resistor, $R = 1\,\Omega$, is positioned on the ground line of the power supply to the FPGA board. During power measurements, both voltage and current are recorded simultaneously. The Teledyne LeCroy HDO6104A-MS High Definition Oscilloscope [158] measures the potential difference across terminals $V_{\text{RS+}}$, $V_{\text{RS}-}$, and V_{IN}, connected via the Power Sense connector. The current through the shunt is obtained by measuring the voltage drop, enabling calculation of power using the following formula, as $V_{\text{RS}-}$ is at the ground and the shunt resistor is 1Ω:

$$P = (V_{\text{IN+}} - V_{\text{RS+}}) \times V_{\text{RS+}}$$

4.2.6. Results

Each experiment measured power consumption across 45,270 samples for top-1 accuracy to assess the effectiveness of sponge poisoning. The findings reveal that this method can be used not only to measure power but also to potentially improve model accuracy. Certain crafting techniques yielded higher accuracies, presenting an efficient real-time training alternative for FPGA accelerators. By tuning weight configurations, pre-trained accelerators can be enhanced without requiring resource-intensive backpropagation, providing a streamlined approach to model improvement.

4.2.6.1. B-DNN-Accelerator Crafted Weights

In the B-DNN-Accelerator weights are either 1 or 0, and a total of 16,711 neurons were identified as inactive neurons i.e. 0. The zero weights in each layer were adjusted to high values to identify neurons that could be activated ("firing neurons (FNs)") without causing the accuracy to fall below 80%. This threshold is defined based on the B-DNN's

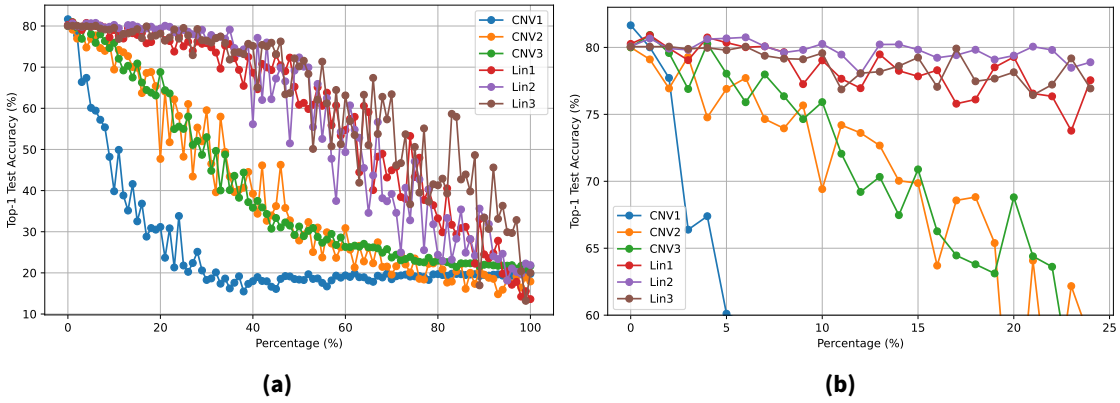


Figure 4.15.: E1 Layer-Wise EvoWeights Optimization: Demonstrating the percentage distribution for higher power consumption while maintaining top-1 test accuracies above the threshold. (a) 0-100%. (b) 0-25%.

baseline top-1 test accuracy of 81.65%, measured across 45,270 test samples, allowing for a maximum drop of only 1.65% in accuracy.

The flipping of weights is aimed to achieve higher `xnorpopcount` operations. This increase in `xnorpopcount` operations is primarily driven by achieving higher matches generated through the `xnor` operation, which directly leads to increased `popcount` operations. This process involves summing bits via `bitwise` operations and `carry` propagation. As the number of matches increases, more `bitwise` and `carry` operations are required, generating additional computational activity. These increased operations cause more transitions between 0 and 1 within the hardware, as the matched 1's from the `xnor` operation propagate through the circuit (such as through adders, registers, and accumulators). This propagation demands more power to charge and discharge the circuit capacitance (C), resulting in higher dynamic power consumption. For instance, in `xnorpopcount` operations, when a consistently high `popcount` is achieved (e.g. 7 out of 8 bits are set to 1), multiple bits toggle during addition, which significantly increases switching activity. This heightened switching leads to more ripple effects in the adder and registers processing these results, causing a notable rise in dynamic power consumption. Dynamic power is directly proportional to the switching activity (α), meaning more frequent transitions increase the energy requirements. Additionally, the cumulative effect of these transitions across Increased parallelism or enhanced coordination between processing units in the FPGA accelerator may introduce additional overhead, particularly in terms of control logic and synchronization, which can further elevate power consumption.

$$P_{\text{dynamic}} = \alpha \cdot C \cdot V^2 \cdot f$$

Figure 4.15a illustrates the progression of 0 to 100% activated FNs across layers with their corresponding top-1 test accuracies for E1. Figure 4.15b provides a focused view of 20% FNs from Figure 4.15a, highlighting the optimal percentage of neurons that maintain accuracy above the set threshold. Similarly, Figure 4.16 depicts firing neuron activation percentages and accuracies for experiments E2 to E15. The results highlight that convolutional layers exhibit greater sensitivity to weight crafting, with the first convolutional layer being

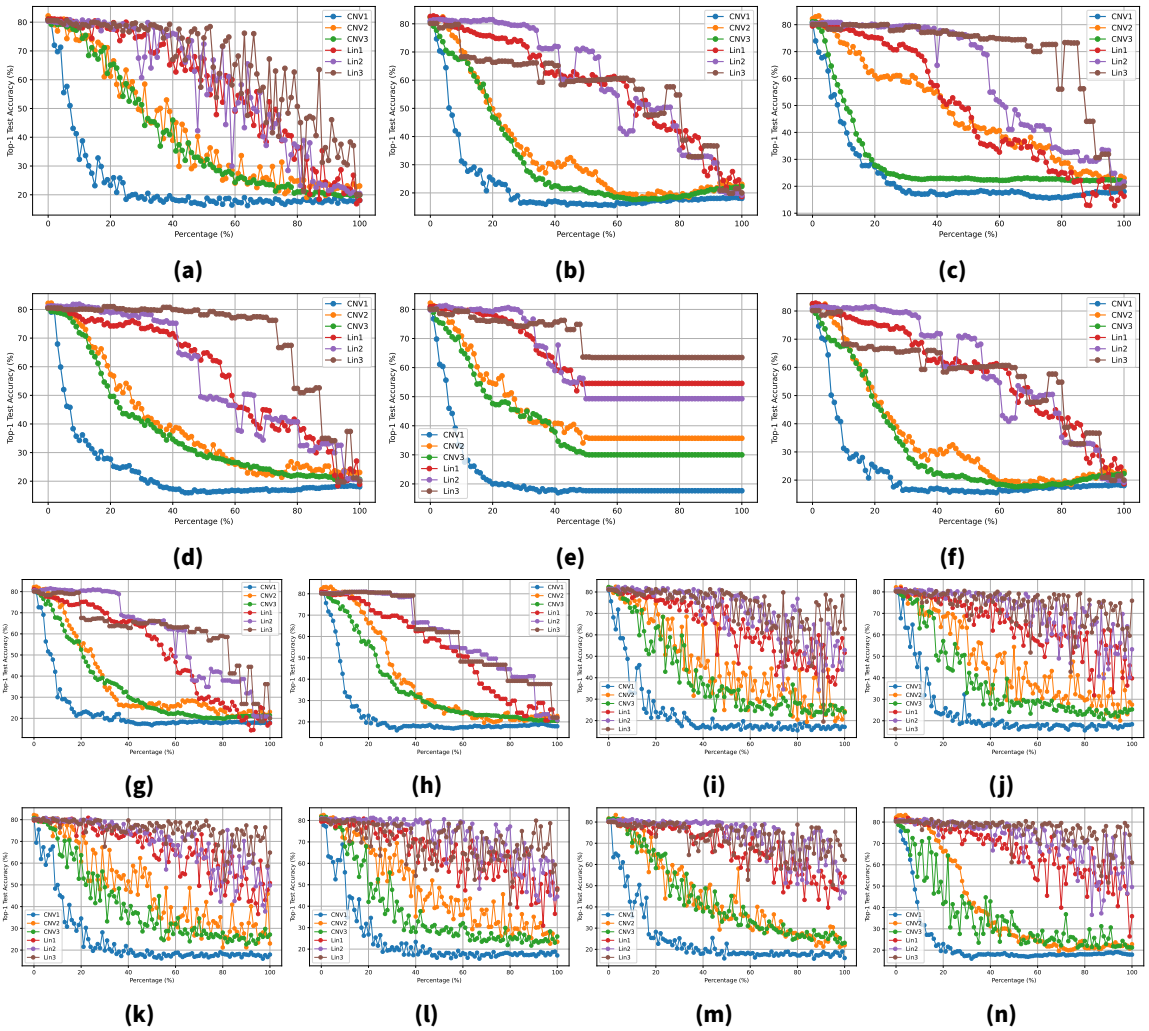


Figure 4.16.: Layer-Wise EvoWeights Optimization: Demonstrating the percentage distribution for higher power consumption while maintaining top-1 test accuracies above the threshold. (a) E2. (b) E3. (c) E4. (d) E5 (e) E6. (f) E7. (g) E8. (h) E9. (i) E10. (j) E11. (k) E12. (l) E13. (m) E14. (n) E15.

particularly susceptible. This finding suggests that models with a higher proportion of linear layers are more favourable targets for sponge poisoning, as linear layers consistently demonstrate lower sensitivity to crafted weights. In certain techniques, these layers tolerated a higher percentage of inactive neurons transformed into FNs without significantly affecting accuracy.

The sponge poisoning led to an increase in power consumption and a drop in the final top-1 test accuracy of each technique alongside the impact on FPGA-Accel Throughput, Inference Time (IoAccel) and FPGA-Accel Runtime is illustrated in Figure 4.17. All methods demonstrated a detrimental rise in B-DNN power usage and other key performance metrics alongside a slight accuracy drop. E9, which applied quadrant-wise firing of neurons, exhibited the highest power increase, with maximum and mean power consumption increases of 17.41% and 16.0%, respectively, while only a minor accuracy drop of 1.06%. This technique also caused a 2.31% increase in inference time, along with a comparatively

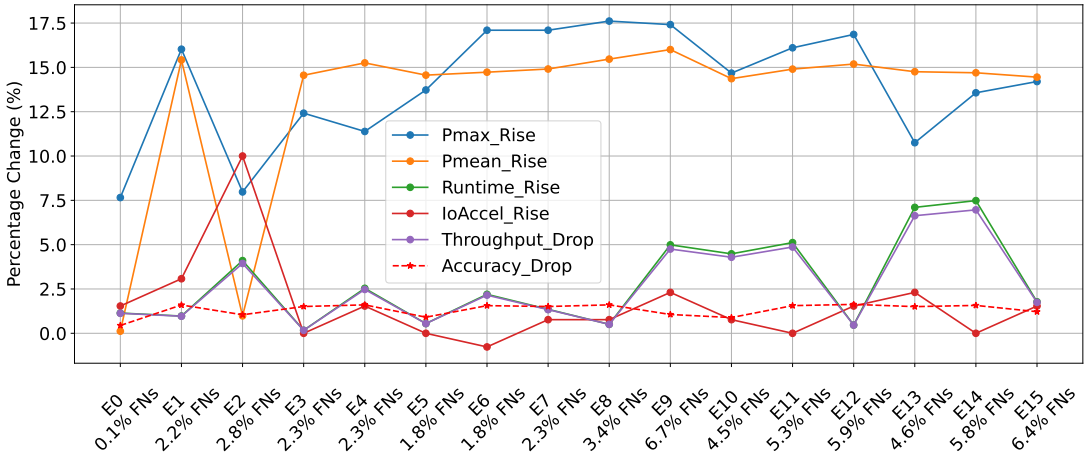


Figure 4.17.: Impact of Sponge Poisoning on System Performance Across all Techniques: Power Rise, Runtime Rise, Inference Time (IoAccel) Rise, Throughput Drop and Accuracy Drop

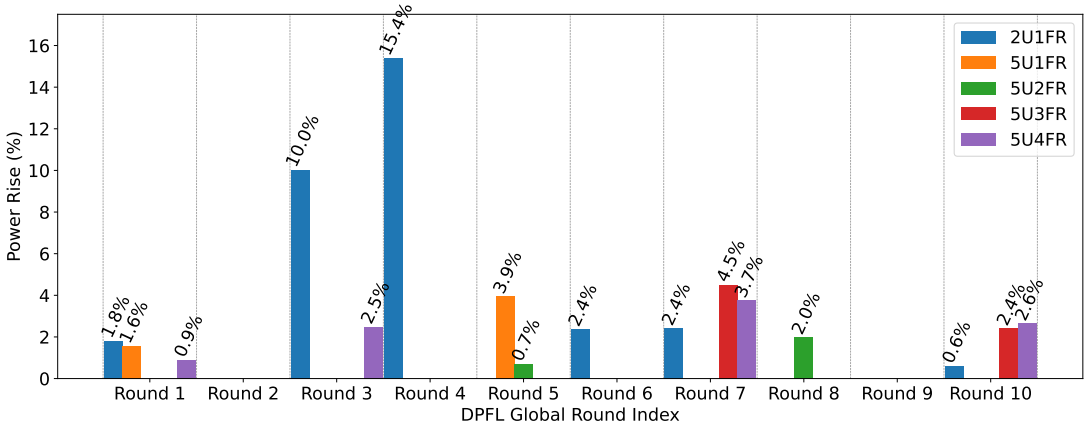


Figure 4.18.: Impact of Sponge Poisoning on Power Rise During DP-Secure FL Global Rounds

larger rise in runtime (4.99%) and a notable reduction in throughput (4.75%). E9 also showed the highest percentage of firing neurons (FNs) at 6.7%, equivalent to 1,122 out of 16,711 inactive FNs summed across all layers of B-DNN.

These findings also suggest that power consumption can be significantly amplified in models with non-binary weights and activations, as such models rely on GEMM matrix multiplication rather than the highly power-efficient `xnorpopcount` operation used in B-DNNs. Despite the high number of firing neurons, the B-DNN's power increase remained relatively modest at 1.174x, partially due to its compact architecture with only three convolutional and three linear layers. For sponge poisoning to be more impactful, models with a higher neuron count, greater layer depth, and more linear layers would be advantageous, given the comparatively lower sensitivity of linear layers to weight crafting. Consequently, architectures with a higher proportion of linear layers may be more susceptible to power-focused attacks, as they consistently exhibit reduced sensitivity to weight modifications.

4.2.7. Section Discussion

The highly effective quadrant-wise flipping method for EvoWeights, illustrating high power consumption and having a reasonable impact on system performance metrics, was evaluated within a DP-Secure FL setup featuring fine-tuning at both the server and user levels. The DP-Secure FL setup included ten global rounds, where the server aggregated weights from honest users alongside fixed-crafted weights from free-riding users in each round. These aggregated weights (PoisWeights), containing sponge-poisoned contributions, were subsequently distributed back to honest users for inference. The study examined five DP-Secure FL configurations: a two-user setup (2U1FR) with one free-rider and a five-user setup (5U1FR to 5U4FR) where the number of free-riders increased from one to four while maintaining at least one honest user. The power consumption of the honest user was measured using the experimental setup detailed in Section 4.2.5.

Figure 4.18 illustrates the percentage increase in maximum power consumption for an honest user when utilizing PoisWeights, compared to the baseline power consumption observed when all participants in the DP-Secure FL setup operate with clean, honest weights. This demonstrates an elevated power consumption in almost all DP-Secure FL configurations and global rounds, bypassing existing security and privacy FL protocols. For instance, in 2U1FR, the power increase in the 4th round neared 16%. In particular, even setups with minimal free-riders, such as 2U1FR and 5U1FR, exhibited a significant power increase. Such power surges can significantly affect edge devices, particularly those operating near peak capacity or deployed in sensitive and critical applications where consistency and reliability are essential. Even a single global round's power rise can have lasting consequences, as honest users continue to perform sponge-poisoned inference until the subsequent global round. This prolonged impact increases power consumption and degrades critical performance metrics, including latency, throughput, and system stability. These effects underscore the potential of sponge poisoning attacks in DP-Secure FL to exploit system vulnerabilities, bypass security protocols including fine-tuning, manipulate system resources, disrupt operational integrity, and amplify risks for real-time and reliability-critical applications.

The reduced impact in later rounds is attributed to the use of fixed-crafted weights throughout all rounds. This strategy is designed to prevent additional weight crafting computations during each round, accommodating the resource constraints of edge devices and ensuring that the free-rider does not incur extra computational overhead. However, the diminishing effect of the attack over successive rounds can be mitigated by dynamically crafting adversarial weights in each communication round of DP-Secure FL to align EvoWeights with the continuously evolving global model updates, further amplifying the attack's effectiveness.

This work presents a novel free-riding attack strategy in DP-Secure FL. Initially, all users actively participate in collaborative learning, but the Free-Rider deviates from honest collaboration, submitting one-time malicious EvoWeights instead of genuine updates. EvoWeights are generated through a lightweight evolutionary algorithm that optimizes layer-wise weight manipulation using mutation and crossover operations. The generation

process includes 16 distinct mutation experiments, ranging from manual and automated flipping strategies, ensuring layer-wise optimal performance.

This study addresses the shortcomings of traditional model replacement and gradient poisoning attacks, which often fail in DP-secure FL due to reliance on crafting sponge examples and costly back-propagation. By exposing vulnerabilities in FPGA-based DP-Secure FL systems with fine-tuning enabled, our approach uses EvoWeights to achieve 6.7% neuron activation and up to 17.41% increased power consumption, while limiting accuracy reduction to only 1.06%. Additionally, it negatively impacts other performance metrics, including a 2.31% rise in inference time, a 4.99% increase in runtime, and a 4.75% reduction in throughput. The attack significantly elevates power consumption and computational costs at both the Global Server and honest participant ends across nearly all global rounds, leading to higher temperatures, potential denial of service, system crashes, and reduced device lifespan. Existing privacy and security measures, such as DP, secure aggregation, fine-tuning, outlier detection, and sanitization methods, are insufficient against this free-rider's EvoWeights sponge-poisoned attack. Our findings reveal that architectures with more neurons, deeper layers, and a higher proportion of linear layers are particularly vulnerable due to their reduced sensitivity to weight manipulation.

For future work, we consider more complex bit-flipping methods, which consider an abstract model of the switching activity. Furthermore, with feedback from real-time hardware power monitoring and reinforcement learning, more effective crafted weights could be generated. This approach can then be extended to also consider the updated weights during each global round instead of using fixed-crafted weights. While these ideas will certainly improve the effectiveness of the attack, EvoWeights already pose a serious threat in DP-Secure FL. As a potential countermeasure, we are investigating if an honest server can initiate injected watermarks, designed to be robust against DP and security settings in DP-Secure FL. Although challenging to implement, watermarks can detect malicious free-riders and enhance the defence mechanisms of FL frameworks against covert, resource-draining attacks.

5. Side-Channels for Countermeasures

5.1. Remote Identification of Neural Networks by Power Fingerprints

The work described in this chapter was published in [5] and is joint work with co-authors Dennis Gnad, Michael Hefenbrock and Mehdi Tahoori.

Neural network fingerprinting has emerged as an important tool in digital forensics to detect model piracy [159], [160], [161], [162], [163], [164]. Typically, such methods exploit characteristics of an NN, such as its decision boundary, to uniquely identify a model [159]. Extracted fingerprints can serve as initial evidence in piracy investigations and also enable detection of model manipulations, including retraining, compression [160], or quantization [164].

However, existing fingerprinting approaches are limited in applicability. They often assume access to model outputs or decision boundaries, making them unsuitable for MLaaS scenarios or cloud-based FPGAs, where models are accessed as black boxes. Moreover, these methods are typically restricted to classification tasks and may reveal sensitive information about the training data.

To overcome these limitations, this work proposes a novel fingerprinting method based on power side-channel information. Instead of relying on functional behavior, an abstract model of the power consumption of NN accelerators is utilized, which can be measured using FPGA-integrated sensors [26]. This approach enables remote fingerprint extraction without requiring access to outputs or training data. The proposed method is robust across different hardware implementations while remaining sensitive to model modifications.

The main contributions are as follows:

- First power-based fingerprinting method for Binary Neural Network (BNN) accelerators on FPGAs
- Abstract power consumption model capable of detecting subtle differences between NNs
- Generation and evaluation of input challenges that produce distinct power signatures

The remainder of this section is taken directly from [5] and is reproduced verbatim.

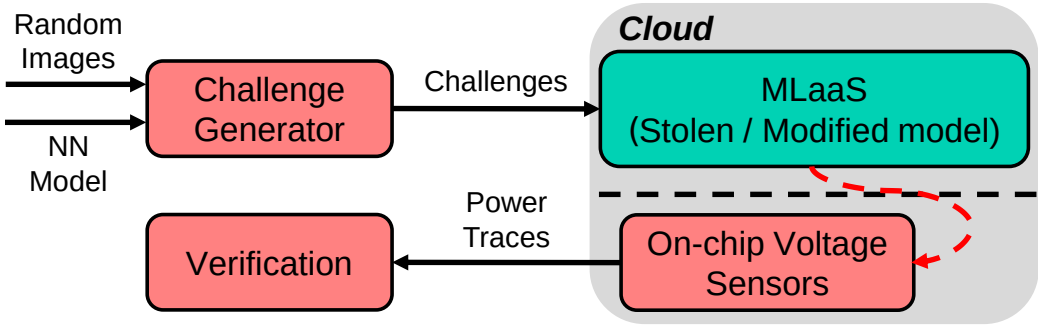


Figure 5.1.: Overview of the challenge generation and identification by the power fingerprint.

5.1.1. Methodology

In the following, we explain our power model and the challenge generation, as well as the verification of the fingerprint. As for criteria, we require the fingerprint to be *unique* to the target model, *effective* with a high matching rate for the target model and *generalizable*, so that it can be applied independent of architecture and dataset. The general flow of our approach is depicted in Figure 5.1. During the execution of the suspect stolen or modified model, voltage fluctuations are measured. From these measurements the power consumption can be approximated and the fingerprint is verified.

5.1.1.1. Power Model

In this work, we utilize an abstract power model $\mathrm{P}(\mathbf{x})$ to optimize inputs $\mathbf{x}$ towards showing specific power consumption. For CMOS based devices, power consumption is high during $0 \rightarrow 1$ / $1 \rightarrow 0$ transitions and low during $0 \rightarrow 0$ / $1 \rightarrow 1$ transitions. We can use this knowledge to approximate the power consumption. In the following, we assume that the accelerator's intermediate states can be reset to 0, so that only $0 \rightarrow 1$ transitions have to be considered.

As we want our power model to be free of features of power traces that are dependent on the implementation, we focus on the accumulated power consumption instead of individual samples. This can be estimated by the multiply-accumulate MAC operations, that implement the matrix vector multiplication $\mathbf{W}\mathbf{x}$ of the vector of inputs $\mathbf{x}$ and weight matrix $\mathbf{W}$, and the number of neuron activations $\varphi(\mathbf{W}\mathbf{x})$.

As we consider BNNs, it is sufficient to sum up the results of the MAC operation and layer outputs for the estimation. The resulting weighted sum is then utilized as the input for the activation function $\varphi(\cdot)$. Hence, the power consumption for a forward pass for a model with L layers and weight matrices $\mathbf{W}^l$ is modelled by

$$\mathrm{P}(\mathbf{x}) = \sum_{l=1}^{L} \|\mathbf{W}^l \mathbf{x}^{l-1}\|_1 + \|\varphi(\mathbf{W}^l \mathbf{x}^{l-1})\|_1 \tag{5.1}$$

with $x^l = \varphi(\mathbf{W}^l \mathbf{x}^{l-1})$ and $\mathbf{x}^0 = \mathbf{x}$ referring to the input of the first layer and $\varphi(\cdot)$ applied element-wise.

5.1.1.2. Challenge Generation

For our approach for challenge generation, we rely on the following hypothesis: ***If a model has sufficient capacity (enough parameters), its behaviour outside the range of the training data is unpredictable***.

Inside the range of the training data, different models with similar accuracy may behave similarly, as they should produce the same output specified by the dataset. However, outside of the region of the data, assuming enough model capacity (parameters), the model may behave arbitrarily [165]. The outputs are not constrained by training data in these regions. Hence, effects such as the initialization may mostly determine the model outputs here [166]. We try to leverage this assumed unpredictability for the generation of our challenges. This is done by formulating an optimisation objective to find a set C of effective and unique challenges for a given model. As a first characteristic motivated by our hypothesis, a challenge $\mathbf{c} \in C$ should be sufficiently different from $x \in \mathcal{D}$ in the training data. This may be measured by

$$\lambda_{\text{oor}}(C) := \sum_{\mathbf{c} \in C} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\| \mathbf{c} - \mathbf{x} \|_2^2 \right], \tag{5.2}$$

in words, the sum of the expected distances (measured in euclidean norm) of the challenges to the inputs $\mathbf{x}$ from the training data set $\mathcal{D}$. Evidently, high values of λ_{oor} should produce challenges *outside of the range (oor)* of the training data. Consequently, our hypothesis should take effect, and $\mathrm{P}(\mathbf{x})$ for such challenges should vary between models.

Additionally, the power consumption for the challenges in C should be noticeably different from those of the training data $\mathcal{D}$. To assess and encourage this property, we define

$$\lambda_{\text{Pdiff}}(C) := \mathrm{P}_{\text{max}} - \mathrm{P}_{\text{min}}, \tag{5.3}$$

with $\mathrm{P}_{\text{min}} := \min_{\mathbf{c} \in C} \mathrm{P}(\mathbf{c})$ and $\mathrm{P}_{\text{max}} := \max_{\mathbf{c} \in C} \mathrm{P}(\mathbf{c})$, and

$$\lambda_{\text{Pclust}}(C) := -\sum_{\mathbf{c} \in C} \min\{ |\, \mathrm{P}(\mathbf{c}) - \mathrm{P}_{\text{min}} \,|, |\, \mathrm{P}_{\text{max}} - \mathrm{P}(\mathbf{c}) |\}. \tag{5.4}$$

For higher $\lambda_{\text{Pdiff}}(C)$, the set of challenges spreads over a higher range of power, while higher values of $\lambda_{\text{Pdiff}}(C)$ (note the negation), $\mathrm{P}(\mathbf{c})$ for the challenges should cluster closely around either P_{min} or P_{max}, depending on which is closer.

Finally, to achieve a diverse set of challenges in C and avoid a collapse through $\lambda_{\text{Pclust}}(C)$, we introduce

$$\lambda_{\text{div}}(C) := \frac{1}{|C|} \sum_{(\mathbf{c}, \mathbf{c}') \in C \times C} \| \mathbf{c} - \mathbf{c}' \|_2^2, \tag{5.5}$$

to assess the diversity of the challenges in C.

Having defined these terms, we can now state our optimisation objective for finding a set C for a given model as,

$$\underset{C}{\text{maximize}} \quad \lambda_{\text{oor}}(C) + \lambda_{\text{Pdiff}}(C) + \lambda_{\text{Pclust}}(C) + \lambda_{\text{div}}(C). \tag{5.6}$$

By maximizing the (weighted) sum of these terms, we find a set of challenges C satisfying the desired properties.

Since some of the terms exhibit a different scale and pose trade-offs, we additionally introduce coefficients $\alpha_{(\cdot)} \in \mathbb{R}^+$, to scale the different terms and increase their contribution to the overall objective. As the objective is differentiable (almost everywhere), the optimisation may be solved using gradient-based optimisation, e.g., Adam [93], starting from a random set of challenges C. To respect the range of valid inputs to the network, e.g., $[0, 255]$ for pixel values, the iterates are projected (clipped) to the respective feasible range after each update step. The calculation of $\lambda_{\text{oor}}(C)$ may be especially expensive for large $\mathcal{D}$. Thus, a Monte Carlo approximation for the expected value

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\, \|\mathbf{c}-\mathbf{x}\|_2^2 \right] \approx \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{c}-\mathbf{x}^{(n)}\|_2^2 \text{ with } \mathbf{x}^{(n)} \sim \mathcal{D} \tag{5.7}$$

with N samples may be used in each step of the optimisation (similar to mini batching in neural network training).

5.1.1.3. Response Verification

In the following, we denote a positive / negative challenge as an input that creates high or low power consumption, respectively. Valid challenge-response pairs of the target model should have distinguishable power consumption measurements. Thus, for verification of the fingerprint we need to classify the responses to challenges as positive or negative. For this, we set a boundary between the maximum and minimum values that should separate negative challenges from positive challenges. We calculate the matching rate as the correct classifications and divide it by the total, for instance 128. Additionally, we define the matching rate gap (MRG) as the difference between the matching rate for the correct model and the maximum matching rate among the incorrect models.

5.1.2. Experimental Setup

For our experiments we generate NN accelerators with the publicly available FINN framework [31] and load them on a Zynq Ultrascale ZCU104. Experiments are conducted on two devices, of which one is placed in a shielded climate chamber (Weisstechnik LabEvent T/210/40/EMC [96]) for evaluation of the extracted fingerprint under different temperatures. The investigated architectures are an MLP with 64 neurons in each layer (MLP-64) and a VGG-like architecture [128] to represent CNNs. These architectures are provided in

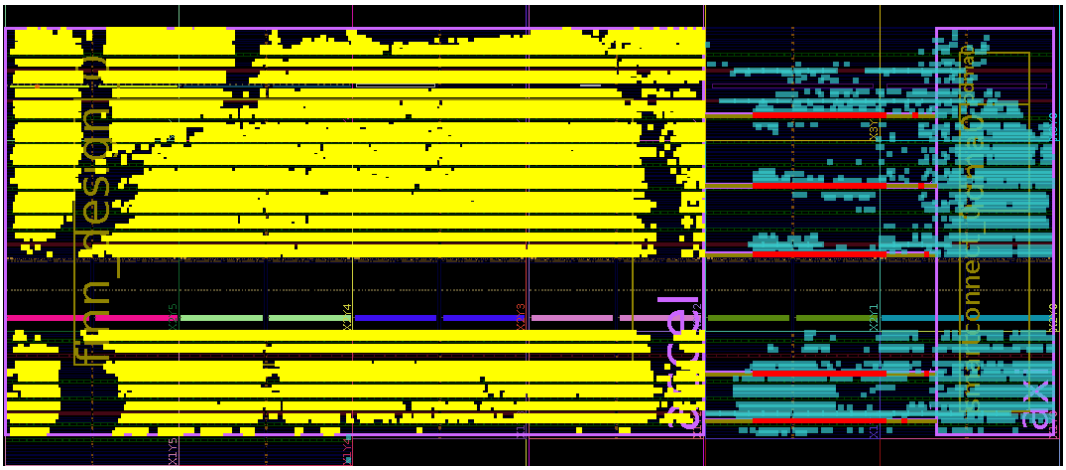


Figure 5.2.: Floorplan of an accelerator (yellow) and sensors (red)

the Brevitas repository [34]. All models have binary weights and activations. The MLPs are trained on the MNIST dataset [90] and the VGG-like models on CIFAR-10 [129].

We generate 128 challenges for each model with an The used initial step size is 0.001 and a period of 5000 iterations for a step size decay of 0.1. The number of Monte Carlo samples to estimate λ_{oor} in each step is set to 128. We initialize the challenges randomly, run the optimization for $50,000$ iterations and save the best result.

Between each inference, the intermediate states of the accelerator's logic are brought to a low state by querying the challenge with lowest modeled power, which sets as many intermediate states as possible to 0. This form of reset is required, as it is an assumption in our power model and in a remote scenario we do not have permission to actually reset the device. Thus, we run a total of 256 inferences for a fingerprint extraction. For each measurement we repeat the inference 20 times and then calculate the average power trace.

We present an exemplary floorplan for one of the MLP-64 accelerators in Figure 5.2. The accelerator and 5 sensors are mapped on the same device, but on different logic/clock regions. For our results we take the average of those 5 sensors. The accelerator runs at 50MHz and the sensors at 100MHz.

In total we conduct 200 experiments, cross-checking 5 MLP-64 models and 5 VGG-like models at different temperatures and across two devices. Additionally, we evaluate the influence of different temperatures on our method. The maximum allowed temperature for the chosen devices is 45°C.

We further process the power traces to filter out unnecessary information and improve our classification results. We isolate the parts of the traces where there is a large gap between positive and negative samples by setting a threshold of minimal absolute distance that is required between them.

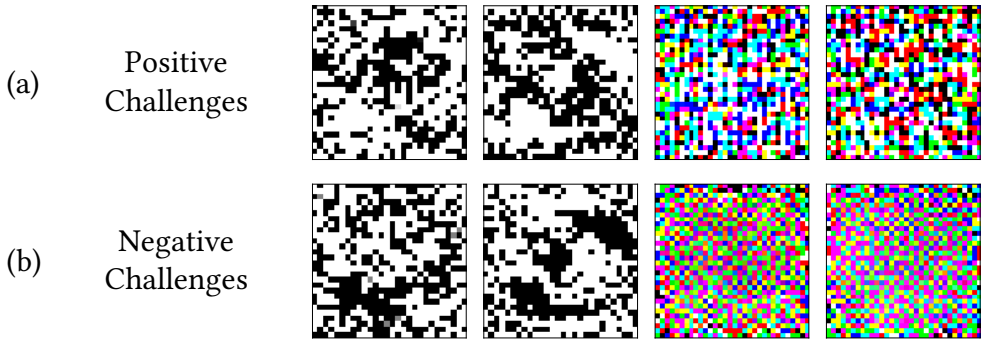


Figure 5.3.: Examples of positive (a) and negative (b) challenges

5.1.3. Results

5.1.3.1. Model Evaluation

Before starting the experiments on actual hardware we evaluate the generated challenges on the algorithmic model level. We present an example for a positive and negative challenge in Figure 5.3 for 2 MLPs and for 2 CNNs. Here, it already becomes visible, that for different models our approach generates very different challenges. We also note that, at least for gray-scale images, some areas the positive and negative challenges are reversed. This indicates that these areas influence the resulting high and low power consumption.

Next, we show the distribution of modeled power consumption in Figure 5.4 for 2 models. For the target model for which the challenges have been originally generated, the power splits up in two groups. This is caused by the λ_{Pdiff} and λ_{Pclust} terms of our loss function. Here, we see that the effect only applies for the correct model and not the incorrect one. This supports our initial hypothesis, that different models (with different initialization) behave differently and unpredictable for *oor* data.

The effect of λ_{oor} is best visible for the VGG-like model trained on CIFAR10, as the dataset is more complex than MNIST. In Figure 5.5 we present a comparison of average pixel distance to images in the original dataset when α_{oor} is set to 0 and when α_{oor} is set to 1000. Blue bars show the average pixel distance of images from the dataset to each other and orange bars the pixel distance of challenges to the CIFAR10 images. In our experiments, setting this term to 0 also removes the uniqueness from the fingerprint.

5.1.3.2. Preliminary Experiments

In our initial evaluation, we first select a total of 128 samples from the MNIST data-set. The selected samples are divided into two sets, with one set containing 64 samples with the highest modeled power consumption, while the other set contains 64 samples with the lowest modeled power consumption. Our objective is to investigate whether this simplistic approach is adequate for the power fingerprint.

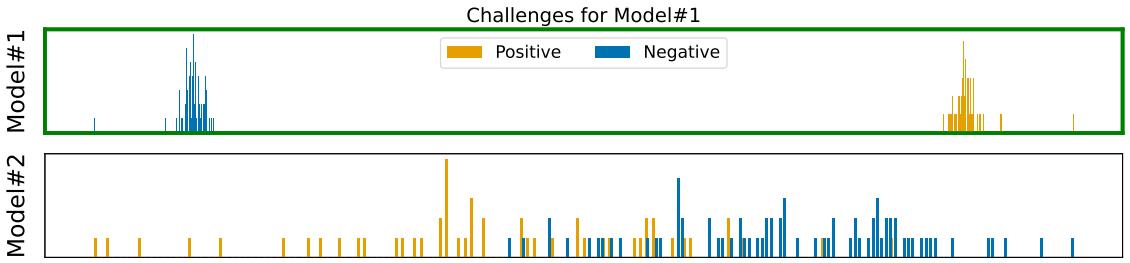


Figure 5.4.: Histograms of modeled power consumption for two models with challenges generated for model#1.

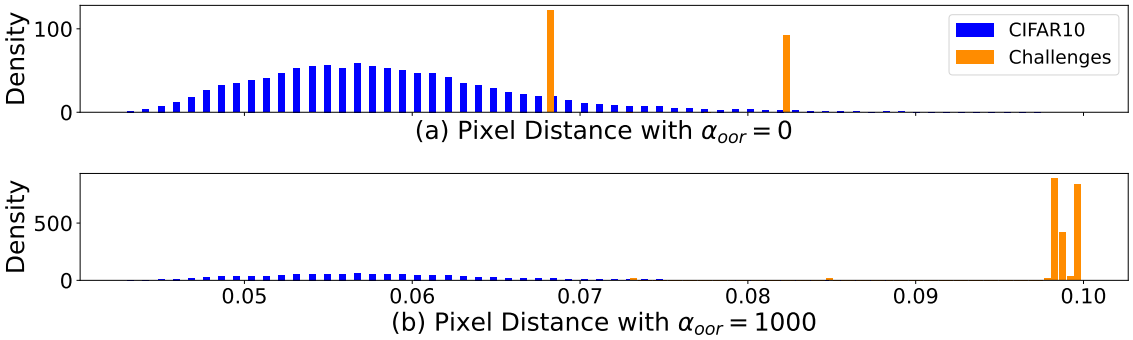


Figure 5.5.: Comparison of challenges generated with $\alpha_{\text{oor}} = 0$ affecting the pixel distance and with $\alpha_{\text{oor}} = 1000$.

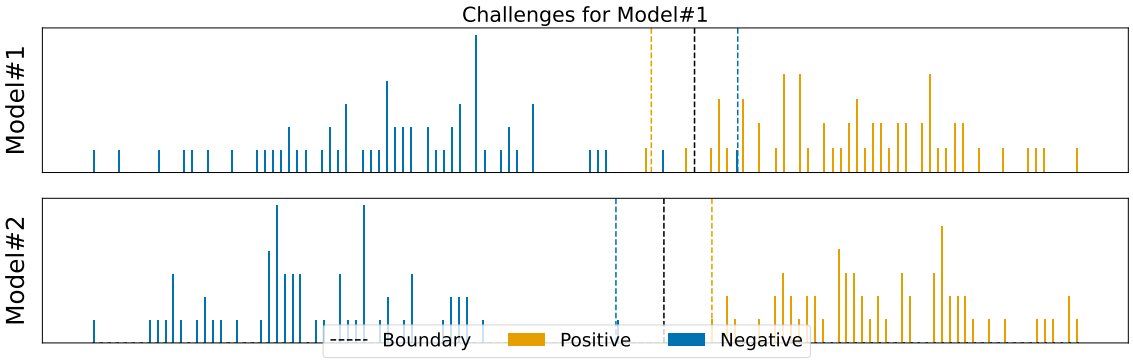


Figure 5.6.: Verification with chosen MNIST samples shown for challenges of model#1.

We proceed by evaluating the performance of this approach by running challenges for model#1 on 5 different models, while measuring the power consumption. Our analysis revealed that the power consumption caused by the challenges sampled from the dataset, are not unique to the target model and thus, fail to meet our requirements for the fingerprint. Thus, the samples from the original dataset fail to meet our requirements for the power fingerprint. The results for 3 of these evaluations are depicted in Figure 5.6. The yellow and blue dotted lines represent the minimal positive challenge and maximal negative challenge, respectively, while the decision boundary, as described in subsubsection 5.1.1.3, between them is denoted by a black dotted line.

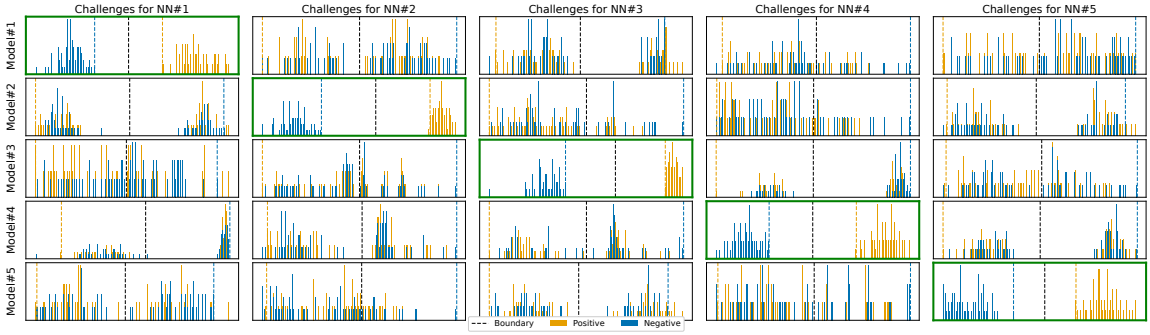


Figure 5.7.: Visualization of accumulated power consumption on the MLP-64 with the correct models identified on the diagonal.

5.1.3.3. Fingerprint Extraction

In the following, we evaluate the generated challenges for fingerprint extraction for 5 models of each architecture. We present an example for how the power consumption is distributed for correct and incorrect models and then show the matching rates for all of the 200 experiments.

The boundary for matching challenges and responses is defined as in subsubsection 5.1.1.3, with responses above the boundary declared as positive and responses lower than the boundary declared as negative. In Figure 5.7 we show the histograms of responses for the MLP-64 models. All generated challenges are queried on all models, with the models varying along the rows and the challenges along the columns. Thus, the correct model for the matching challenges can be found on the diagonal, marked by a green border. The yellow and blue dotted lines represent the minimal positive challenge and maximal negative challenge, respectively, while the decision boundary, as described in subsubsection 5.1.1.3, between them is denoted by a black dotted line. It is clearly visible, that for the correct models, positive and negative responses split up just as expected. For incorrect models, the responses are scrambled.

We go on by calculating the matching rate for all experiments and present the results in Figure 5.8. Along the diagonal, which is the correct model for the respective challenge, we reach matching rates of 100% in almost all cases. For the incorrect models, the matching rates are relatively close to 50%. This huge gap between correct and incorrect models allows us to set a lower threshold for the amount of challenges that need to pass. Only at 0°C and 45°C for the VGG the matching rate drops to 91% and 95% respectively. This is due to environmental temperatures leading to offsets in the power traces [167]. We can safely set the threshold to 90% and reach 100% identification accuracy without any false positives.

In Table 5.1 we further summarize these results. With the threshold set to 90% matching rate we get 100% True Positive Rate (TPR) and 0% False Positive Rate (FPR) in all cases. In general, the MRG is lower for the VGG-like model but still large enough to effectively distinguish between a correct and incorrect model. We also notice that, while the matching rate might drop for varying temperatures, the average MRG remains stable with variations

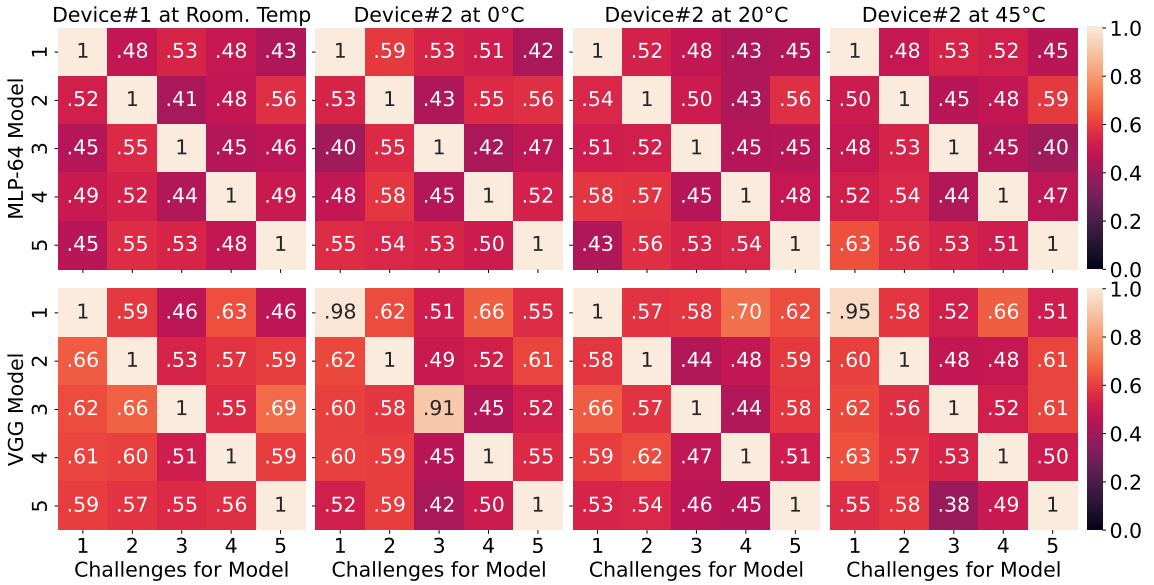


Figure 5.8.: Matching rates of challenges for two different architectures on 5 models respectively.

Table 5.1.: True / False Positive Rate (TPR / FPR) for evaluated architectures, with matching rate threshold set to 90%.

	MLP-64				VGG			
Metric	20±2°C	0°C	20°C	45°C	20±2°C	0°C	20°C	45°C
TPR / FPR	1./0.	1./0.	1./0.	1./0.	1./0.	1./0.	1./0.	1./0.
Avg. MRG	0.46	0.43	0.45	0.44	0.36	0.36	0.39	0.37
Min. MRG	0.44	0.41	0.42	0.37	0.31	0.31	0.30	0.29
Max. MRG	0.48	0.45	0.48	0.47	0.41	0.41	0.46	0.42

of only 0.02. Thus, it can be concluded that the fingerprint of a model can be safely extracted, even in an unknown environment. The results of our fingerprint method can compete with current methods [159], [160], and moreover, the power fingerprint does not require reading any form of output from the accelerator. Furthermore, our method could be applied to any kind of task and is not limited to classification. We would also like to add, that the chosen architecture does not have a strong impact on the method, as it mainly depends on the power consumption of the MAC units and activations. We show that this works for fully connected as well as convolutional layers in our experiments.

5.1.4. Section Discussion

The results of our experiments show that the generated challenges are effective for fingerprint extraction independent of the evaluated NN architecture. We can safely set a requirement of 90% matching rate to achieve 100% identification accuracy on the correct models. Additionally, there is a significant gap between the matching rate for the correct

model and an incorrect one, allowing robust identification with no false positives. We show that our method is robust to chip-to-chip and temperature variations, up to the maximum allowed environmental temperature of 45°C. In the future we are planning to extend our research with external measurements and more platforms and devices. Overall, fingerprints of NN accelerators are a powerful tool in the detection and prevention of model piracy, compression, and modification.

5.2. Side-Channel Aware Neural Network Training

The work described in this chapter was published in [6] and is joint work with co-authors Dennis Gnad, Jonas Krautter, Martin Gotthard and Mehdi Tahoori.

Remote side-channel attacks pose a significant threat to the confidentiality of data processed by DNN accelerators. This section presents a training-based countermeasure that aims to reduce observable side-channel leakage while preserving the intended functionality of the model.

The approach follows a two-step methodology. First, the input reconstruction attack from *Power2Picture* [4] is reproduced on a realistic platform based on the Zynq UltraScale+ MPSoC. Subsequently, the training process of the victim network is modified to explicitly account for the leakage exploited by the attack. This enables the model to maintain its classification performance while reducing the information available through side channels.

The effectiveness of the proposed method is evaluated by reapplying the side-channel attack to both the original and the protected models, demonstrating a reduction in attack success.

The remainder of this section is taken directly from [6] and is reproduced verbatim.

5.2.1. Methodology

In the proposed approach, we adapt the training of the victim classifier to fulfill two training objectives at the same time:

(i) Perform the original image classification as intended, meaning that the model is able to solve the underlying problem with high accuracy.

(ii) Integrate the model's leakage as loss into the training process, minimizing the victim classifier's leakage through the side-channel.

Since the exact way of how the input and the weights of the victim network influence the measured trace is not known in our scenario, we cannot directly optimize the weights of the victim network to minimize the leakage. In technical terms, we cannot back-propagate through the side-channel, since it is unknown if this function is differentiable.

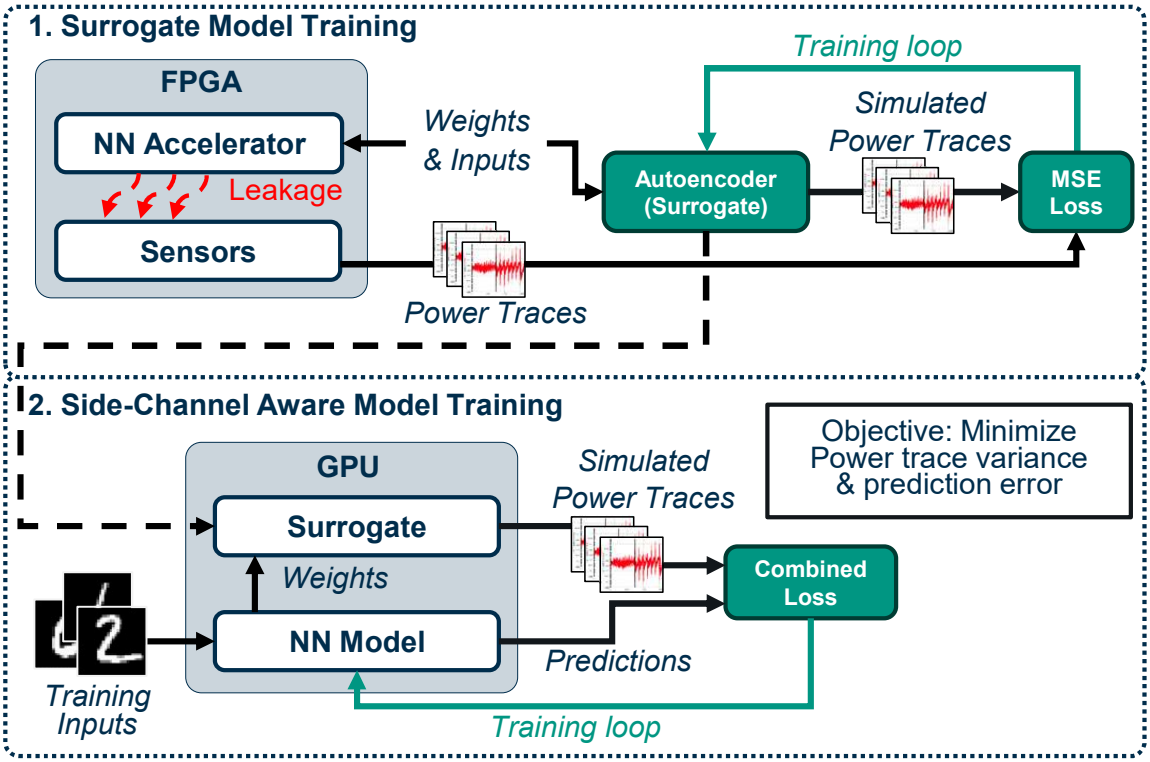


Figure 5.9.: An overview of the surrogate model training and the following side-channel aware training. Instead of using the side-channel, the surrogate model is used to simulate power.

Our solution to address this problem is to implement and train a *surrogate model*, which gets the image and weights of the victim network as input, and output the trace this configuration would produce on the real FPGA. This model is differentiable and, thus, closes the gap between weights/images and the resulting power trace. For the training of the victim classifier, we first pass an image to the classifier and get its result. Then we take the original image and the weights of the classifier and pass them as input to the surrogate model, which then outputs the corresponding power trace.

Our objective is to force a constant output of the surrogate model for all inputs we use. For example, this can be either an all-zero output, or the average of all traces. The surrogate model itself is trained beforehand and is not modified when retraining the victim network to minimize its leakage. In fact, since our objective is to update the weights of the victim network and these weights are part of the input for the surrogate model, we optimize for input and not for weights in the surrogate model. The loss then gets combined with the loss of the image classification. An illustration of the setup is shown in Figure 5.9.

In summary, our method is implemented as follows:

(i) Generate a training data set consisting of a large variety of image/weights combinations and their real world trace using the unprotected victim classifier.

(ii) Train a surrogate model that outputs a trace given an image and the weights of the victim network.

(iii) With the help of the surrogate model, train the victim network to perform the original classification task, while having minimized information leakage.

To support this setup, opposed to running the FINN NN Accelerator in *const* memory mode, as also presented in [4], we operate it in *decoupled* mode. In that mode, the weights can be changed at runtime without reprogramming the entire design. Overall, we use the same type of board (Xilinx Ultrascale+ ZCU104 Board), and similar neural network. Next to our countermeasure results, we report results for both the const and decoupled setup.

5.2.1.1. *Result Metrics*

To evaluate the effectiveness of the attack, we use result metrics as follows:

Reclassification Accuracy

As mentioned before, the test output of the generator consists of images recovered from the traces of the test data set. We can feed these images once more through the original (victim) classifier and measure the portion of images that were classified correctly. We call this metric *Reclassification Accuracy*. The closer the recovered images are to the original images, the more likely the classifier will classify them correctly. Thus, a higher reclassification accuracy is indicative of a more successful attack.

Average Pixel-Level Distance (APLD)

To calculate the APLD, every recovered image is compared to its original by computing the average pixel-wise difference. The more similar the recovered images and the corresponding originals are, the smaller the pixel-level distance is. Ideally, for an attacker, there is no difference, resulting in a pixel-level distance of zero.

Mean Structural Similarity Index (MSSIM)

The MSSIM was developed by Wang et al. [111] and is a metric to evaluate the preservation of structural information in two images. It follows directly from the definition of the *Structural Similarity Index* (SSIM). Since our original images are highly structured, this metric provides useful insight into the quality of our results. The SSIM compares three aspects: luminance, contrast, and structure of both images. The MSSIM is defined by computing the SSIM multiple times. Here, the SSIM is not calculated for the entire image at once. In fact a sliding window is moved over the image, calculating the SSIM in every step. These values get averaged afterwards. For our results, we use a window size of 11. To get a single value for the data set, we simply average all MSSIMs pair-wise.

Table 5.2.: Reclassification accuracy, Mean Structural Similarity Index (MSSIM) and Average Pixel-Level Distance (APLD) for an accelerator without and with applied countermeasure.

model	attack success metrics		
	reclassification accuracy	MSSIM	APLD
ideally secure	10% (guessing)	close to 0.0	close to 255
perfect attack	100%	1.0	0.0
const (baseline)	65.75%	0.56	26.73
decoupled	38.70% (*-29.8%*)	0.34 (*-39.3%*)	32.19 (*+20.4%*)
countermeasure	22.07% (*-66.4%*)	0.25 (*-55.4%*)	35.15 (*+31.5%*)

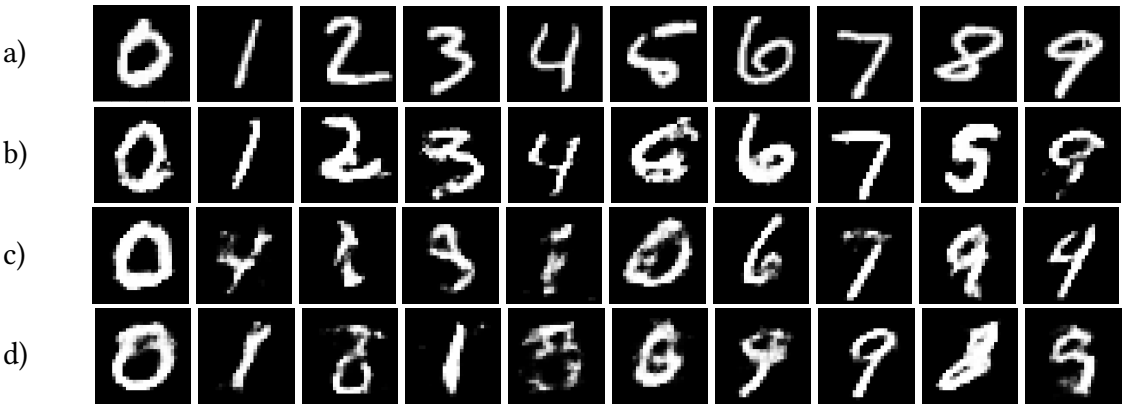


Figure 5.10.: Examples of (a) Original MNIST images, (b) recovered images in *const* memory mode (c) recovered images in *decoupled* memory mode, and (d) recovered images with the countermeasure applied.

5.2.2. Results

Overall, the countermeasure has shown to reduce the amount of information that is recoverable through the chosen side-channel, while maintaining a high classification accuracy. The results are summarized in Table 5.2. The unprotected baseline approach with *const* memory mode reaches a reclassification accuracy of 65.75%. By applying the decoupled mode, we can already achieve results below the baseline, which is due to the additional streaming of the weights hiding some of the leakage. For our training-based countermeasure the *reclassification accuracy* dropped significantly down to only 22.07%. This drop means that the adversary could not restore the images with the same accuracy, indicating that our protection countermeasure effectively reduces the attack feasibility. This is also confirmed by the trend of the *MSSIM* and *APLD* metrics. Furthermore, it is remarkable that the initial classification accuracy of the protected neural network has only suffered a minimal loss with less than 1% accuracy degradation to the unprotected model. This is very important since we do not want our countermeasure to largely affect the primary task of the accelerator.

5.2.3. Section Discussion

Overall, our countermeasure has shown to reduce the attack success by about 43%. With the countermeasure, the victim is able to improve its privacy by making the attack more difficult for the attacker with negligible losses in classification accuracy. As per our design, the countermeasure does not need additional hardware resources or runtime for inference, since we only adapt the weights of the network. However, we were not able to completely prevent information being leaked through the side-channel. Besides the need for more research on this specific topic, our countermeasure is a complement in an existing security concept and not a standalone solution to prevent side-channel leakage.

6. Side-Channels for Functional Safety in AI Accelerators

6.1. Out-of-Distribution Detection Using Power-Side Channels for Improving Functional Safety

The work described in this chapter was published in [9] and is joint work with co-authors Michael Hefenbrock, Mahboobe Sadeghipourrudsari, Dennis Gnad and Mehdi Tahoori.

Despite their strong performance, NN accelerators suffer from a critical limitation: the inability to reliably detect inputs outside their training distribution, i.e., OOD inputs [65]. Such inputs often lead to incorrect predictions, posing serious risks in safety-critical applications. Ensuring a *fail-safe* behavior, where unreliable predictions are detected and flagged, is therefore essential. Existing approaches, such as confidence thresholding [65], [168], entropy-based methods [67], and Bayesian NNs [169], address this problem but introduce significant limitations.

In particular, many methods rely on computationally expensive procedures or require access to inputs, outputs, or intermediate activations, delaying detection until after inference. This added latency is problematic for real-time and resource-constrained edge deployments, especially for Quantized Neural Networks (QNNs). Consequently, there is a need for concurrent OOD detection mechanisms that operate directly on hardware during inference.

To address this, this work proposes a novel method based on power side-channel analysis. By leveraging on-chip FPGA voltage sensors, power traces are collected during inference and used to characterize normal (ID) behavior. Differences between ID and OOD power profiles are then exploited using simple scoring functions to detect anomalous inputs.

This approach enables non-intrusive, concurrent OOD detection without requiring access to model internals. Furthermore, the method can be implemented directly in hardware with negligible overhead and zero additional latency, making it particularly suitable for safety-critical NN accelerator deployments.

The main contributions are as follows:

- Non-intrusive and concurrent power side-channel based OOD detection for QNN FPGA accelerators

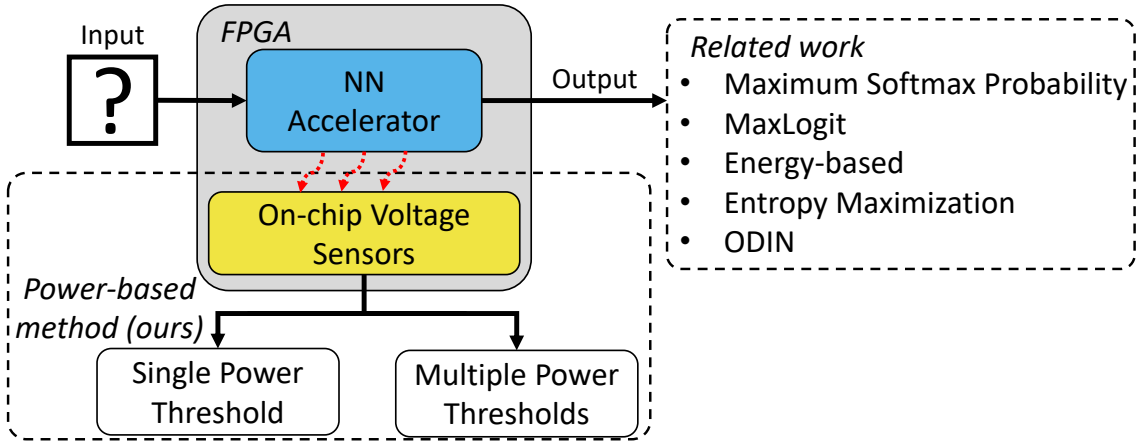


Figure 6.1.: Setup for our OOD detection using on-chip voltage measurements and related work using outputs or internal values of the neural network.

- Zero-latency and low-overhead hardware implementation of the detection mechanism

The remainder of this section is taken directly from [9] and is reproduced verbatim.

6.1.1. Concurrent OOD Detection

In this section, we present our method for utilizing the power side-channel of QNN FPGA accelerators for OOD detection. Our approach for the detection mechanism, relies on calculating a power profile of expected power consumption for ID data. The power profile can be represented as a single or multiple thresholds τ_i.

Figure 6.1 shows the general setup for the power side-channel based OOD detection. On-chip sensors measure voltage during the inference, and a score function f_{score} maps the trace to a single value. If the score crosses a threshold (or multiple), we set the OOD flag. The steps of our method can be summarized as:

1. Collect power measurements from the training or test set
2. Offline analysis of power traces and determine threshold(s)
3. Write threshold(s) to memory and evaluate on hardware

We assume that the traces are represented as fixed-point numbers in hardware, which leads to the same results in the software evaluation with float-32. This is due to the simple calculations, which do not require high precision and the sensor values being represented as integer values. The fractional part is only required for taking the average of sensors.

We evaluate established criteria [65], such as the FPR-95, receiver operating characteristic (ROC) curve and area under ROC curve (AUC). The ROC curve illustrates the balance between true positive rate (TPR) and false positive rate (FPR) at various decision thresholds

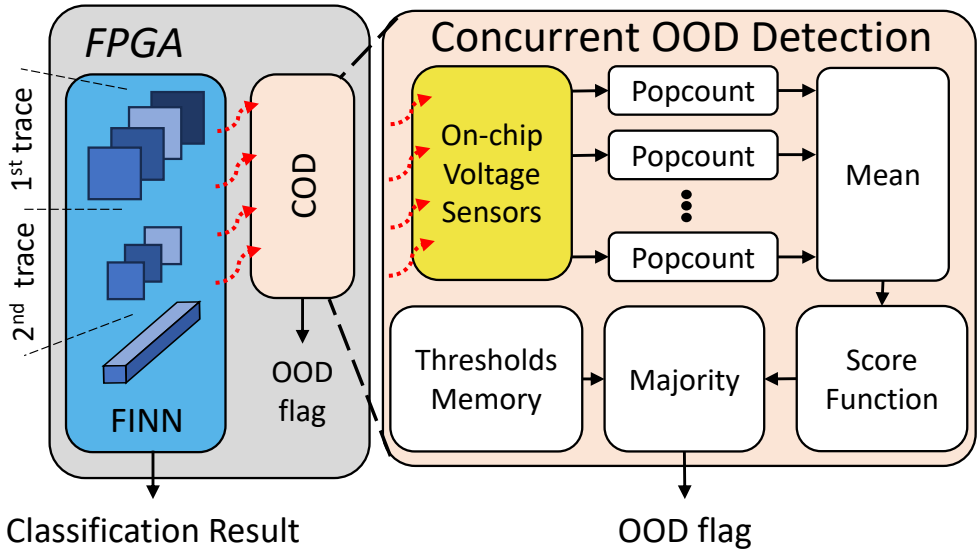


Figure 6.2.: Overview of hardware implementation of power side-channel based COD on FPGA with FINN.

and the AUC is a summary statistic. The FPR-95 is the FPR of OOD examples at 95% TPR of ID examples. Lower FPR-95 indicates better OOD detection performance.

6.1.1.1. OOD Detection Hardware Implementation

To ensure the efficiency of the proposed technique, the design is implemented together with the FINN accelerator IP on the FPGA. The score of a sensor value v_i for the one clock cycle is denoted as

$$score = f_{\text{score}}\left(\frac{\sum_i^{\#\text{sensors}} \text{Popcount}(v_i)}{\#\text{sensors}}\right) \tag{6.1}$$

and the detection mechanism for a single threshold is defined as follows, where τ is the threshold

$$f(score) = \begin{cases} \text{accept}, & \text{if score} \leq \tau \\ \text{reject}, & \text{otherwise} \end{cases} \tag{6.2}$$

We implement Eq. 6.1 and Eq. 6.2 as displayed in Figure 6.2. This figure shows the FINN accelerator and the hardware implementation of the power side-channel OOD detection mechanism. The RDS are started with the start of the inference, and stop after a configurable number of clock cycles. In each clock cycle the sensors measure voltage fluctuations as an array of '1's and '0's. We utilize `popcount` modules, which count all '1's with a combinational module and pass them to `mean`. The module `mean` calculates the average of all popcount outputs. These two modules work simultaneously within a single clock cycle. The average sensor result is passed to the `score function`, which maps the power trace to a single value. Here, we explored simple computations, such as summation or minimum. For the case of multiple thresholds, each section of the trace must be compared to different thresholds. As these thresholds can be written to memory

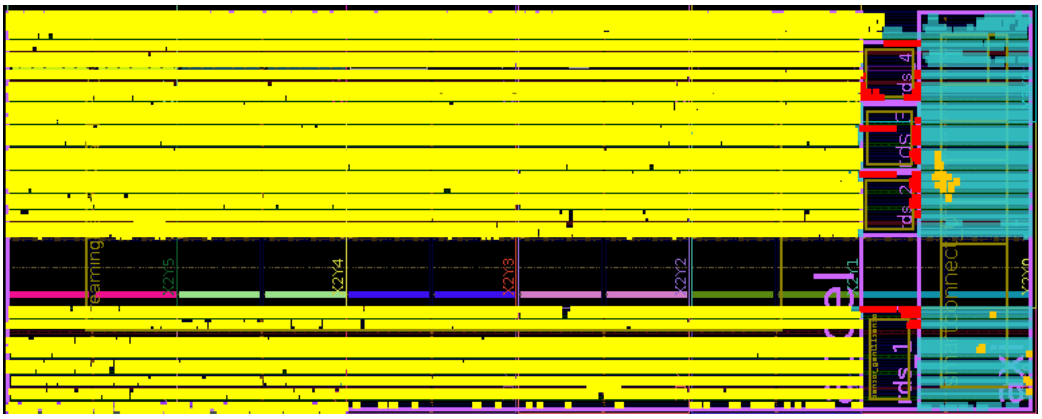


Figure 6.3.: Floorplan of the FINN accelerator (yellow) and multiple RDS (red), OOD detection (orange), remaining control and debug logic (cyan).

during runtime, there is no need to reconfigure the device. The `majority` module performs the comparison and tracks the results for each section and issues the OOD detection flag after the last trace is processed.

6.1.2. Experimental Setup

We employ the publicly accessible FINN framework [31] to generate all neural network accelerators for our experiments. Subsequently, we deploy these accelerators onto a Zynq Ultrascale ZCU104. RDS [45] are used to measure voltage fluctuations. These sensors are co-located on the same FPGA. The floorplan for the setup is displayed in Figure 6.3, with the accelerator in yellow, sensors in red and the detection mechanism in orange. In our experiments, the sensors run at 200MHz and the accelerator at 50MHz, however, the sensors can also run at higher speeds with the correct calibration. The frequency of the sensors is thus not a limitation.

We evaluate our method on a multi-layer-perceptron with 4 layers and 64 neurons in each layer, which we will call MLP-64. VGG-like [128] CNNs with 6 convolutional layers, followed by 3 fully-connected layers, are also considered for the GTSRB [170] and CIFAR-10 dataset. The classification accuracy of the MNIST, CIFAR-10 and GTSRB model are 92.95%, 84.61% and 98.06% respectively. The models are quantized to 1-bit weights and activations with the Brevitas [34] library. We use the following four OOD datasets for models trained on MNIST [90]: (1) FashionMNIST [91], (2) CIFAR10 [129] (grayscale, resized to 28×28), (3) MNIST-C [171] and (4) independent Gaussian noise drawn from $\mathcal{N}(0, 1)$.

Each of these datasets consists of 10k samples. The MNIST-C dataset consists of various corruptions applied to all sample of the original MNIST dataset. Some of the corruptions, such as change in brightness, do not have an effect on binary NNs, as the pixel values need to cross a threshold to make a change. Thus, we use only the following corruptions from the MNIST-C dataset: *rotate*, *line*, *dotted_line*, *zigzag*, *inverse*, *stripe*, *canny_edges*, *impulse_noise*. For the CNNs, we use the following OOD datasets: (1) TinyImageNet [172],

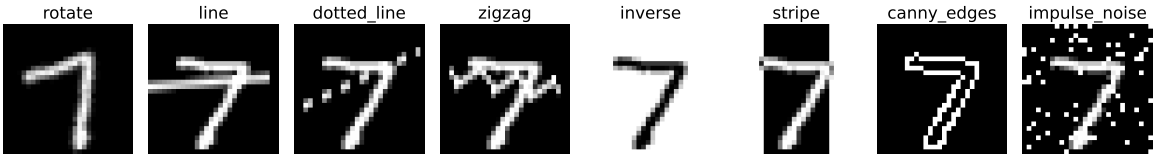


Figure 6.4.: Exemplary samples from MNIST with various corruptions as applied in MNIST-C [171].

(2) SVHN [173], (3) GTSRB / CIFAR-10 (respectively) and (4) independent Gaussian noise drawn from $\mathcal{N}(0, 1)$. For the model trained on GTSRB, we reduce the number of OOD samples to fit the number of ID samples, so that the ROC is not skewed and the FPR95 and AUC remain informative. Here, the number of samples is 3870. We repeat the experiments 20× and find no significant variation in the results.

The intermediate states of the FINN accelerator do not reset after an inference. As the data-dependent power consumption of the accelerator is driven by the switching activity, this makes the power measurements related to an input sample dependent on the previous one. Thus, we can determine our threshold by collecting power traces for transitions from ID to ID samples. Then, if an OOD samples is queried after an ID sample, the score function of the power trace will be remarkably different. A visualization for this behavior can be found in Figure 6.7, which displays the average power trace for MNIST and corresponding OOD datasets. Here, the traces can be easily distinguished by the first voltage drop or the minimum of the trace.

For the comparison with existing methods we use the python library *pytorch-ood* by Kirchheim et al. [174]. We consider *Entropy Maximization*, *MaxLogit*, *MaxSoftmax*, *Energy-based* OOD-detection and *ODIN*. The performance on each OOD dataset for these methods is presented in Figure 6.6.

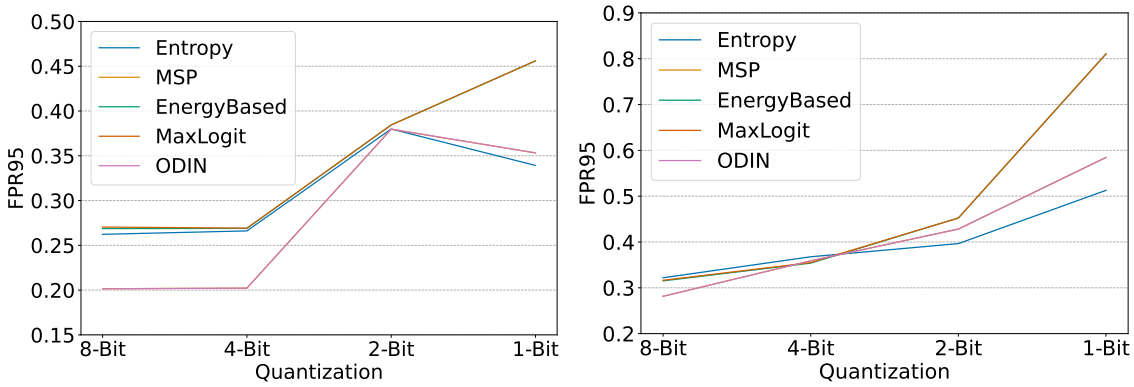
6.1.3. Results

We begin by evaluating related work on QNNs and then report the results for our proposed method. For the comparison with related work, we consider the results from a fixed point implementation, as it would be in hardware.

6.1.3.1. Evaluation of Related Work

As a preliminary experiment, we compare the FPR-95 of the considered related work for different levels of quantization. The results for this experiment are presented in Figure 6.5, which shows the FPR-95 for the MLP64 with 1,2 4, and 8-Bit quantization. We notice that with decreasing precision of weights and activations, the FPR-95 increases, displaying an deterioration of the detection mechanism.

On top of model quantization, the calculations in hardware, like entropy and softmax, are mostly performed with fixed-point representation. Thus, we implement fixed-point variants of the evaluated methods from related work. Here, we consider binary neural



(a) CIFAR10 **(b)** FashionMNIST

Figure 6.5.: Evaluation of FPR-95 for related work with models trained on MNIST on three quantization levels and CIFAR10 and FashionMNIST as OOD datasets.

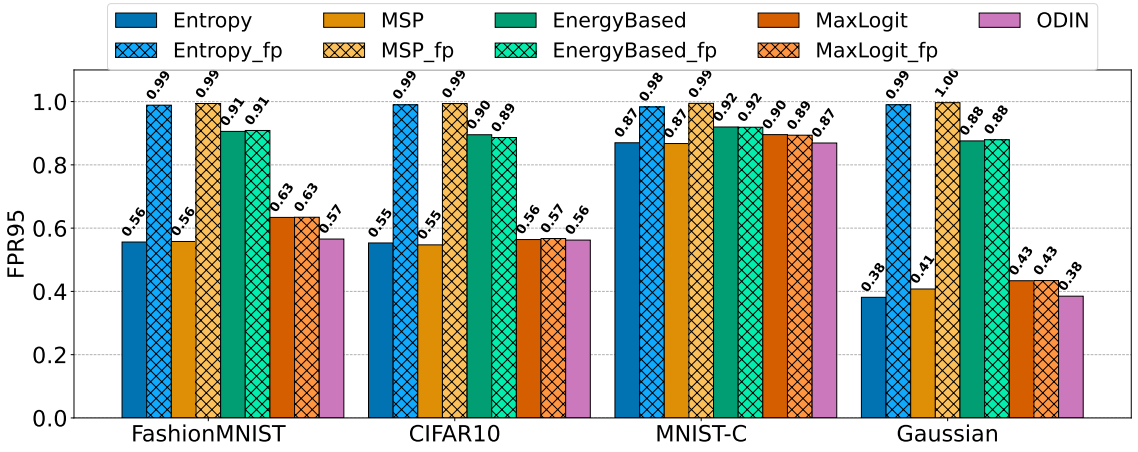
networks. For softmax we also consider an hardware-efficient approximation as in [175]. We use 10-bit fixed-point numbers with 2 bit for the integer and 8 bit for the fractional part, which is sufficient for representing the logits and softmax values without loss of accuracy in any of the models. ODIN [69] is excluded from this comparison, as it requires back-propagation, which cannot be performed efficiently on the FPGA and would also result in 2× latency for the second inference of the perturbed input.

Figure 6.6a to Figure 6.6c show the results of this comparison. In all cases, entropy maximization [67] does not function properly anymore with an FPR95 above 90%. We find that the effect of fixed-point numbers on existing methods is minimal in most other cases, however also breaks MSP [65] for MNIST. Also for complex CNN tasks, detection capabilities of existing methods slightly deteriorate.

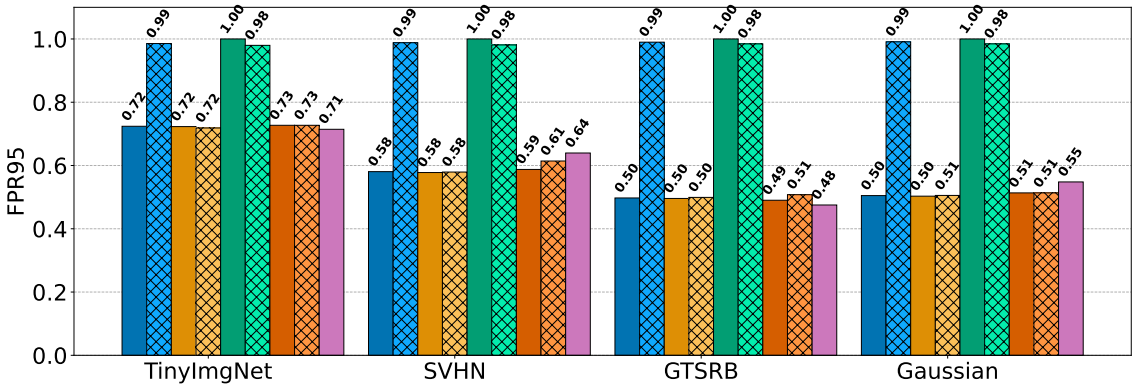
6.1.3.2. Results of our Side-Channel Based COD

Initially, we analyze power traces of the MLP-64 trained on MNIST to evaluate our method on a simple dataset and model. Figure 6.7 displays, how the power traces are significantly different, supporting our assumption that the transition from ID samples to OOD samples can be detected by the power consumption. Here, it is sufficient to determine the threshold by the minimum of the trace, as the first layer creates the most notable voltage drop. For the GTSRB model, we select the sum as the score function with a single threshold. Finally, for the CIFAR-10 model, we use 9 thresholds for the detection and can still get the OOD detection result at around 3/4 of the inference.

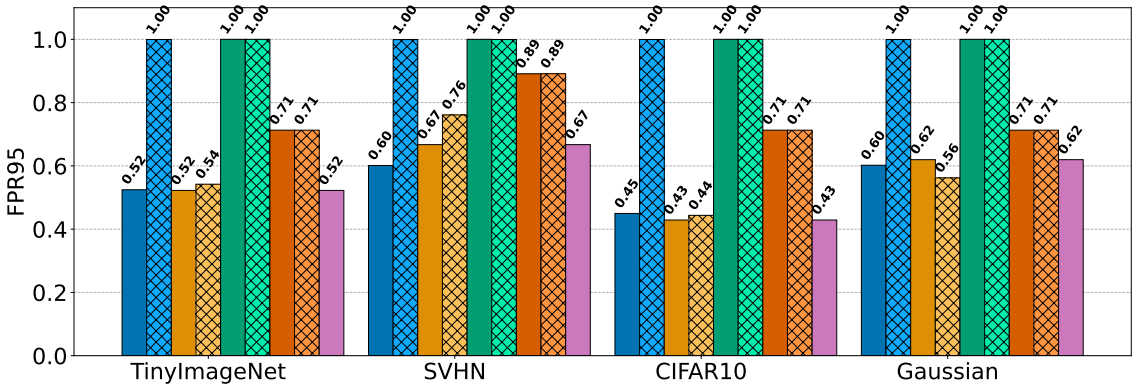
Figure 6.8 shows the ROC for our power side-channel-based OOD detection for the MNIST (a), GTSRB (b) and CIFAR-10 (c) model. We notice that gaussian noise seems to have distinguishable features from the ID sets, independent of model or training dataset. Noise can be perfectly detected in almost any case with our method. As shown earlier in Figure 6.6, this is not possible using methods from related work when considering QNN.



(a) MLP trained on MNIST



(b) CNN trained on CIFAR-10



(c) CNN trained on GTSRB

Figure 6.6.: Comparison of OOD Detection methods with floating point and fixed point on a binary MLP trained on MNIST.

Table 6.1, Table 6.2 and Table 6.3 summarize all of our experiments and show a comparison to related work. Our method beats related work in FPR-95 in almost every case. This method outperforms related work for any of the chosen OOD datasets, noteworthy also MNIST-C, and is also computationally efficient. It is easily implemented on an FPGA as it requires no complex calculations.

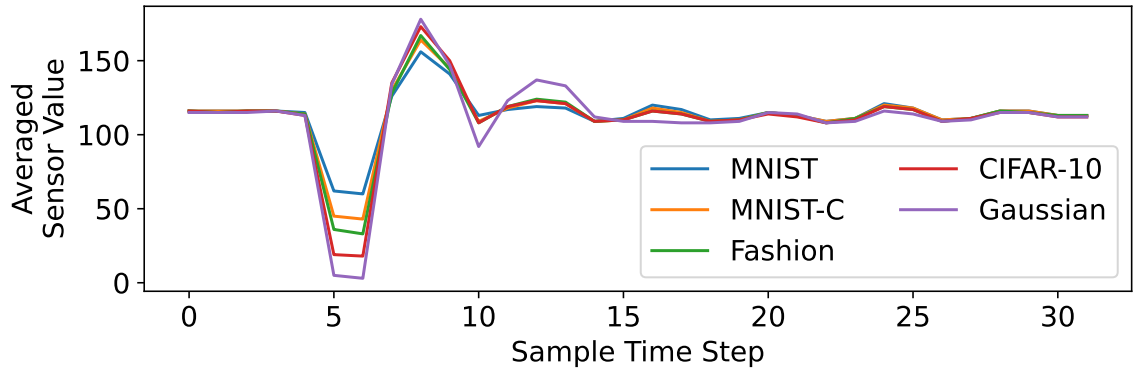


Figure 6.7.: Averaged power traces for each dataset for an MLP trained on MNIST.

Table 6.1.: False positive rate at 95% TPR (FPR-95) and area under curve (AUC) of the OOD-Detection with fixed-point / float32 for MLP-64 (MNIST). ↓ means lower values are better and ↑ means high values are better.

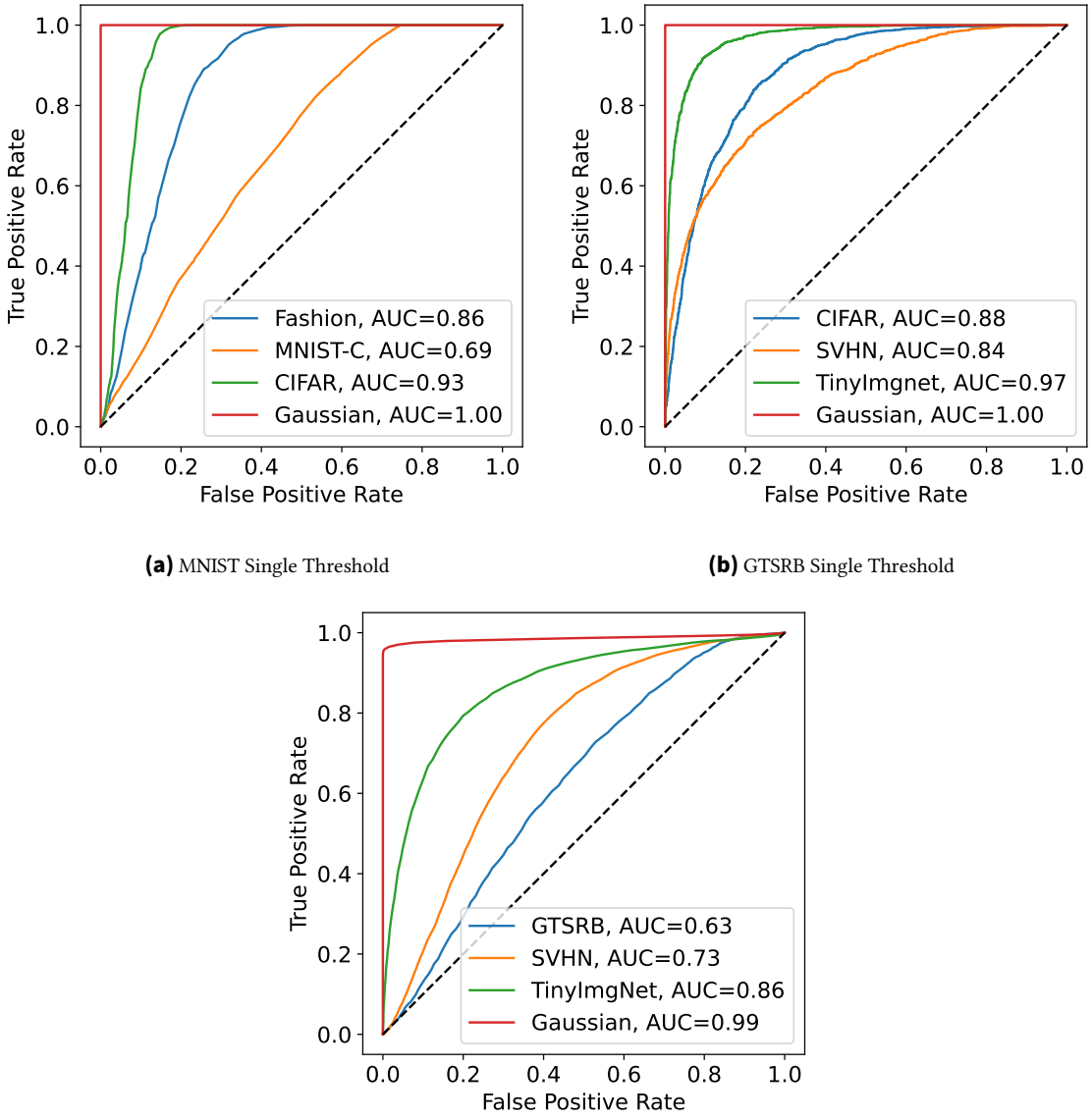
Method (ID=MNIST)	FashionMNIST		MNIST-C		CIFAR-10 (Grayscale)		Gaussian Noise	
	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑
Ours	**0.31**	**0.86**	**0.67**	**0.69**	**0.13**	**0.93**	**0.0**	**1.00**
MaxLogit	0.63 / 0.63	0.85 / 0.85	0.89 / 0.89	0.72 / 0.72	0.56 / 0.56	0.85 / 0.85	0.43 / 0.43	0.88 / 0.88
MSP	0.99 / 0.54	0.18 / 0.85	0.99 / 0.86	0.31 / 0.72	0.99 / 0.54	0.18 / 0.85	0.99 / 0.40	0.15 / 0.88
Entropy Max.	0.98 / 0.55	0.23 / 0.85	0.98 / 0.86	0.43 / 0.72	0.98 / 0.55	0.23 / 0.85	0.98 / 0.38	0.20 / 0.88
Energy-based	0.90 / 0.90	0.58 / 0.59	0.91 / 0.91	0.55 / 0.55	0.88 / 0.89	0.60 / 0.60	0.87 / 0.87	0.61 / 0.62
ODIN	0.56	0.85	0.86	0.72	0.56	0.85	0.38	0.88

6.1.4. Section Discussion

For all the evaluated datasets, we find that the hardest task is detection of samples from the MNIST-C set. These samples are closest to actual MNIST samples, making the detection a more challenging task. Still, we can achieve an FPR-95 of 0.67, an improvement of around 25% from the best result of related work. Yang et al. [176] describe these as near OOD

Table 6.2.: False positive rate at 95% TPR (FPR-95) and area under curve (AUC) of the OOD-Detection with fixed-point / float32 for a CNN (CIFAR-10). ↓ means lower values are better and ↑ means high values are better.

Method (ID=CIFAR-10)	TinyImgNet		SVHN		GTSRB		Gaussian Noise	
	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑
Ours	**0.50**	**0.86**	**0.68**	**0.73**	**0.78**	**0.62**	**0.00**	**0.99**
MaxLogit	0.73 / 0.73	0.76 / 0.76	0.61 / 0.59	0.79 / 0.79	0.51 / 0.49	0.82 / 0.82	0.51 / 0.51	0.81 / 0.81
MSP	0.72 / 0.72	0.75 / 0.75	0.58 / 0.58	0.79 / 0.78	0.50 / 0.50	0.82 / 0.81	0.51 / 0.50	0.81 / 0.81
Entropy Max.	0.99 / 0.72	0.25 / 0.75	0.99 / 0.58	0.19 / 0.78	0.99 / 0.50	0.22 / 0.81	0.99 / 0.50	0.19 / 0.80
Energy-based	0.98 / 1.00	0.28 / 0.50	0.98 / 1.00	0.25 / 0.50	0.98 / 1.00	0.22 / 0.50	0.98 / 1.00	0.22 / 0.50
ODIN	0.71	0.76	0.64	0.77	0.48	0.82	0.55	0.80



(a) MNIST Single Threshold

(b) GTSRB Single Threshold

(c) CIFAR-10 Multiple Threshold

Figure 6.8.: ROC of the OOD detection for different datasets and models

sets, which only have semantic shift in comparison with the ID dataset. We notice that the detection mechanism performs worse on the VGG trained on CIFAR-10, which is likely related to the lower accuracy of the model, which in turn results in a less pronounced power profile for ID samples.

We find that the evaluated existing methods [65], [67], [68], [69], [177] in almost all cases show worse performance compared to our method, even though the AUC is in some cases higher. Furthermore, our proposed detection method can be efficiently implemented in hardware and can be performed concurrently to the detection, as it is sufficient to analyze a part of the trace. Thus, we conclude that our method beats the evaluated existing methods

Table 6.3.: False positive rate at 95% TPR (FPR-95) and area under curve (AUC) of the OOD-Detection with fixed-point / float32 for a CNN (GTSRB). ↓ means lower values are better and ↑ means high values are better.

Method (ID=GTSRB)	TinyImgNet		SVHN		CIFAR-10		Gaussian Noise	
	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑	FPR-95 ↓	AUC ↑
Ours	**0.13**	**0.97**	**0.59**	**0.84**	**0.38**	**0.88**	**0.0**	**1.00**
MaxLogit	0.71 / 0.71	0.91 / 0.91	0.89 / 0.89	0.87 / 0.87	0.71 / 0.71	0.92 / 0.92	0.71 / 0.71	0.91 / 0.91
MSP	0.54 / 0.52	0.91 / 0.92	0.76 / 0.67	0.88 / 0.89	0.44 / 0.44	0.92 / 0.93	0.56 / 0.62	0.91 / 0.91
Entropy Max.	1.00 / 0.92	0.08 / 0.52	1.00 / 0.89	0.11 / 0.60	1.00 / 0.93	0.07 / 0.45	1.00 / 0.91	0.09 / 0.60
Energy-based	1.00 / 1.00	0.16 / 0.81	1.00 / 1.00	0.19 / 0.77	1.00 / 1.00	0.15 / 0.82	1.00 / 1.00	0.19 / 0.81
ODIN	0.52	0.92	0.67	0.89	0.43	0.93	0.62	0.91

Table 6.4.: Hardware utilization and power of FINN, RDS, and the proposed OOD detection mechanism

Module	LUT	Registers	Power(W)
FINN	97508 (42.32%)	123001 (26.69%)	0.929W
RDS	2136 (0.93%)	1404 (0.30%)	0.275W
OOD Detection	60 (0.03%)	37 (< 0.01%)	0.014W

in performance. The suboptimal performance of current methods in OOD detection could potentially be attributed to their lack of adaptation to Quantized Neural Network (QNN), which have a reduced parameter space. The difference in parameters of QNNs might reduce the distinctiveness of the logits and softmax values, which in turn lowers the effectiveness of existing methods. We show that our method still works effectively for the worst case of single bit activations and weights, which have a less pronounced power profile than less quantized networks, due to lower switching activity.

Furthermore, our approach works out of the box and does not rely on any other data than the power measurement and the explicit output label of the accelerator. By solely relying on power measurements, we reduce the amount of data-transfers between ARM-core and FPGA required to detect OOD samples.

Table 6.4 evaluates the efficiency of the proposed COD technique in terms of hardware costs compared to the overall system. As shown in Table 6.4, the needed FPGA resource requirement for OOD detection is less than 1% and 0.3% of the LUTs and registers, respectively. This amounts to only 0.2% of the hardware consumption compared to the FINN accelerator. Furthermore, the power consumption for OOD detection is less than 24% of the total power consumption of the entire system. It should be noted that 20% of this power consumption is due to the higher frequency of the sensors compared to the rest of the system. As mentioned before, our approach is zero-latency as it only requires the configured amount of clock cycles and can finish anytime before the inferences finishes.

We propose an innovative approach to efficiently and effectively detect OOD samples in NN FPGA accelerators, using power side-channel measurements and one or multiple thresholds learned from the ID data. Unlike the other existing methods, our method is not only suitable for quantized NNs as commonly used for edge hardware deployment,

but can also be efficiently implemented within the device and detect OOD samples before inference finishes. The proposed approach can be extended to different domains and datasets and enhances the robustness and reliability of edge accelerators in safety-critical domains, ensuring their safe and secure operation even in the presence of OOD inputs. In the future, we are looking into expanding our experiments and method for other types of NNs, such as transformer models, and tasks beyond image classification.

6.2. Concurrent Fault Detection for Binary Neural Network Accelerators via On-Chip Voltage Monitoring

The work described in this chapter was published in [11] and is joint work with co-authors Mahboobe Sadeghipourrudsari, Dennis Gnad and Mehdi Tahoori.

NN accelerators are susceptible to a wide range of runtime faults affecting both memory and logic components. These faults can originate from permanent sources such as aging or manufacturing defects, as well as transient effects including radiation-induced soft errors or voltage fluctuations [178]. Even single-bit errors in weights or activations can silently degrade model accuracy [179], while the complexity of NNs makes fault propagation difficult to predict.

Existing fault detection techniques face significant challenges. Hardware-based methods such as ECC, redundancy, or Cyclic Redundancy Check (CRC) [72], [180], [181] introduce high overhead and offer limited coverage for logic faults. Software-based approaches, including duplicate inference [73], activation checks [74], or hash-based methods [76], improve flexibility but incur runtime overhead and are restricted to specific fault models. This motivates lightweight, real-time detection mechanisms with minimal system impact.

This work proposes a concurrent fault detection approach based on on-chip voltage fluctuation sensors, specifically delay-line-based Routing Delay Sensors (RDS) [45]. These sensors capture voltage variations caused by changes in switching activity, which are influenced by faults during inference. Building on prior work on power fingerprinting [5], a lightweight classifier is trained on side-channel traces to detect abnormal behavior. The method operates non-intrusively in parallel to inference, requiring no modifications to the NN accelerator.

The main contributions are as follows:

- Fault detection method leveraging on-chip voltage sensors for monitoring functional errors in NN hardware
- Detection of faults affecting both weights and logic components
- Low-overhead, reference-free approach suitable for integration into existing systems
- Operation in a gray-box setting without modifications to the accelerator

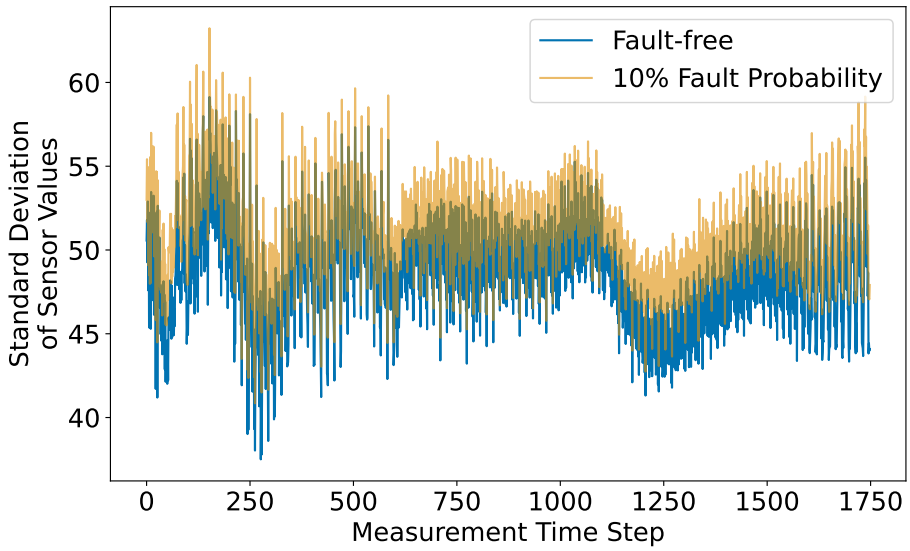


Figure 6.9.: Standard deviation of traces under 10% fault probabilty and fault-free operation.

- Evaluation on PYNQ-Z2 and ZCU104 platforms, demonstrating portability to cloud FPGAs

6.2.1. Voltage Fluctuation Based CED

Our assumption is that runtime faults manifest as measurable deviations in the accelerator's voltage fluctuations, as faults alter the switching activity of the accelerator, which directly affects the dynamic voltage drop. To assess these effects, we collect voltage traces during inference under both fault-free and faulty conditions. Figure 6.9 shows the standard deviation of traces with and without faults, relative to the mean of the fault-free traces. With faults present, traces consistently exhibit higher deviation, confirming that faults induce distinguishable shifts in voltage drops. At lower fault probabilities, the deviation is lower but still significant. Our method utilizes this behavior by analyzing voltage fluctuation measured during NN inference and works as follows:

1. **Fault Injection:** Inject faults via bit flips in model weights or neuron connections.
2. **Trace Collection:** During inference, record voltage fluctuations, producing a high-dimensional trace.
3. **Feature Aggregation:** Divide the trace into fixed-size windows. For each window, the sum of the sensor values is computed to reduce dimensionality.
4. **Thresholding:** Windowed sums are compared against a learned threshold. Each result is weighted and summed.
5. **Decision:** Compare the total score against a global detection threshold to classify the trace as clean or faulty.

This procedure is applied per inference, and the results can optionally be accumulated over multiple inferences for higher confidence. The detector is trained on voltage traces of inferences without faults and with weight bit flips. An overview of the training procedure is given in Figure 6.10.

6.2.2. Classifier Design and Training

Before going into an in-depth description of our method, we define the fault model considered in this work.

Fault Model

We consider two classes of runtime faults: *bit flips in parameter memory* and *stuck-at faults*, i.e., faults in neuron logic.

A bit flip in memory alters the stored weight values during inference. Given a binary weight tensor $W \in \{-1, +1\}^n$, a bit flip at index i inverts its sign:

$$W_i^{\text{faulty}} = -W_i.$$

These faults are injected at runtime with a predefined fault rate $r \in [0, 1]$ indicating the fraction of affected weights.

stuck-at faults (SAF) corrupt the output signals of neurons between layers by forcing them to a constant logic value. Let a_i denote the output of neuron i. A fault sets:

$$a_i^{\text{faulty}}(t) = c \in \{0, 1\}, \quad \forall t,$$

where t indexes the clock cycles. These faults are injected using dedicated fault injection IPs placed between layers, simulating stuck-at behavior in activation signals.

Per-Window Threshold-based Fault Detection

We propose a learnable per-window threshold voting mechanism. Each windowed sum is passed through a steep sigmoid function that compares it to a learnable threshold τ_i:

$$s_i = w_i \cdot \sigma(\alpha \cdot (x_i - \tau_i)),$$

where x_i is the summed sensor value in window i, τ_i is the learned threshold, α is a scaling factor controlling the steepness, and w_i is a learnable vote weight associated with window i. The sigmoid $\sigma(\cdot)$ produces a fault indication, and w_i scales the influence of each window on the final score.

The overall fault score is computed as the sum of all weighted window scores, which is then compared against a global detection threshold to trigger a fault flag. During training,

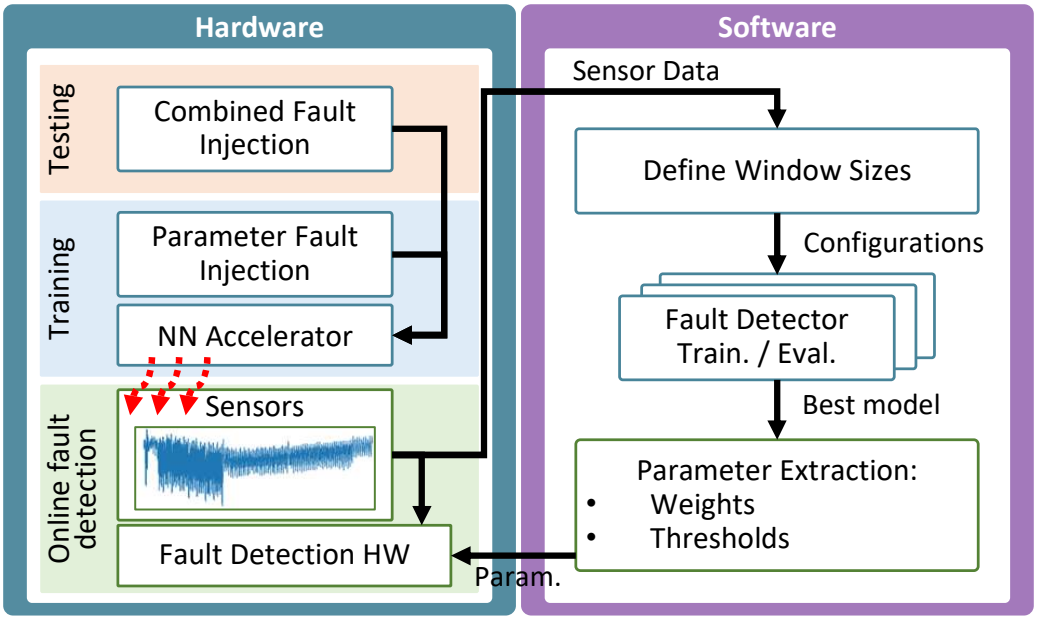


Figure 6.10.: Overview combining hardware and software: Faults are injected into the accelerator, and voltage traces are recorded via on-chip sensors. In software, these traces are used to train the detector. The resulting parameters are written to hardware.

both the thresholds τ_i and the vote weights w_i are optimized using labeled clean and faulty traces.

To ensure efficient hardware deployment, we apply quantized training with constraints on w_i, such that all vote weights are powers of two. This enables replacement of multiplication with hardware-efficient shift operations during accumulation. The thresholds (τ_i) are also quantized to fixed-point format to allow lightweight comparison logic. This results in a fully interpretable and low-overhead detection mechanism that is well-suited for FPGA-based accelerators.

Choosing the window size: The choice of window size plays a critical role in time-series analysis, as it determines the temporal resolution of the features used for detection. We evaluate multiple window sizes, chosen as increasing powers of two and report the range between the smallest window size that achieves stable detection performance and the largest window size that still provides acceptable detection latency.

Detection Over Time

The presence of faults may not always be evident from a single inference, especially under low fault intensities or in noisy environments. Therefore, we accumulate scores across multiple inferences for decision-making. This strategy is particularly well suited for permanent or long-lasting faults, such as bit flips in weight memories, which persist across inferences due to the static nature of stored weights. It also enables the detection of certain transient fault classes, such as burst faults, that affect execution over a short but contiguous

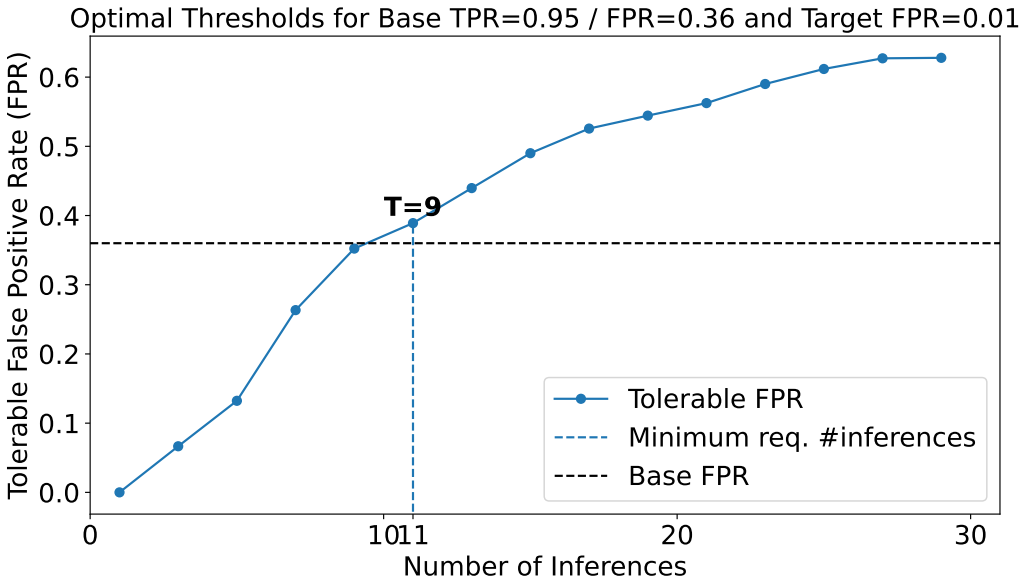


Figure 6.11.: Tolerable FPR as a function of the number of inferences. For a base TPR of 0.95 and FPR of 0.36, a threshold of $T = 9$ over 11 inferences reduces the effective FPR below the target of 0.01.

sequence of inferences. Furthermore, detection over time is particularly advantageous for continuous inference tasks, such as video processing or real-time perception.

Specifically, we define a global threshold T over the sum of individual inference scores across the last N inferences. Figure 6.11 illustrates how the tolerable false positive rate (FPR) increases with the number of inferences for a given true positive rate (TPR). For a base FPR of 0.36 and a target FPR of 0.01, a minimum of 11 inferences with a threshold of $T = 9$ is sufficient to suppress false positives to acceptable levels.

6.2.3. Hardware Design

Our detection method is implemented as a standalone hardware block adjacent to the NN accelerator, as shown in Figure 6.12. The architecture supports a variable number of windows and enables dynamic updates of all detection thresholds and weights at runtime.

Each trace window is accumulated and compared against its corresponding threshold (τ_n^1) using a comparator, effectively implementing a hard sigmoid function. Each vote is multiplied by its weight (W_n) by a shifter, and a simple sequential adder then aggregates the results. The final score is compared to a runtime configurable threshold (τ_2) to generate the fault flag.

The design is parameterizable at synthesis time in terms of the number of windows and their respective sizes. At runtime, threshold values and weights can be written to dedicated memories, allowing flexible adaptation without requiring resynthesis. This

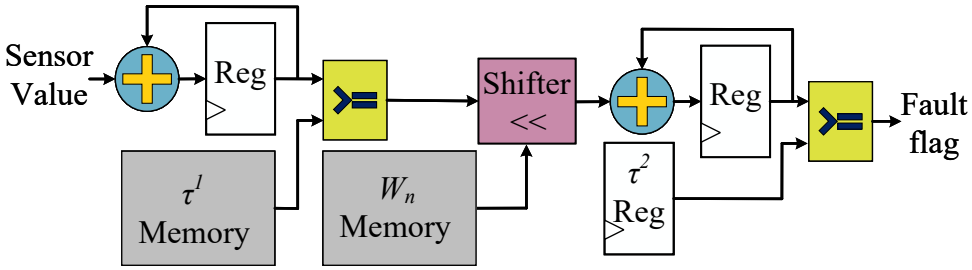


Figure 6.12.: Hardware block for per-window threshold fault detection. Window size and thresholds are fully configurable. Scores are aggregated and evaluated with minimal latency.

Table 6.5.: Summary of evaluated NN models

Model	Platform	Architecture	Accuracy	Throughput (images/s)
MNIST MLP	Pynq-Z2	FC: 784–256–256–256–10	97.18%	180136.74
MNIST CNN	Pynq-Z2	Conv: 16–32–64, FC: 10	97.17%	7781.03
GTSRB VGG	ZCU104	Conv: 6×64, FC: 8	97.98%	6080.79

separation between static architecture and dynamic configuration supports a broad range of deployment scenarios with minimal overhead.

6.2.4. Experimental Setup

For evaluation, we use a Pynq-Z2 (Zynq-7000) and a ZCU104 (Zynq UltraScale+ MPSoC), whose UltraScale+ fabric family is also used in datacenter accelerators. While our method is not limited to FPGAs, this platform provides a flexible prototyping environment for NN accelerators. The approach is general and could be extended to other architectures, such as TPUs or ASIC-based accelerators.

6.2.4.1. Models and Datasets

Table 6.5 summarizes the NN models evaluated in this work. All models are binary, with 1-bit weights and activations, and trained using the Brevitas library [34]. The experiments include a fully connected Multilayer Perceptron (MLP) and a Convolutional NN (CNN), both trained on the MNIST dataset [90], as well as a VGG-like [128] model trained on a subset of the German Traffic Sign Recognition Benchmark (GTSRB) [170]. The MNIST MLP consists of three hidden layers, while the MNIST CNN uses three convolutional layers with increasing filter counts, followed by a fully connected output layer. The VGG-like model uses 6 convolutional layers, with a stride of 2 in every second layer. Setting a stride of 2 replaces max-pool layers without affecting the accuracy.

6.2.4.2. Accelerator Implementation

We use the FINN framework [31], developed by Xilinx Research, to generate hardware accelerators for our binary NN models. FINN maps each layer to a dedicated hardware block using a deeply pipelined dataflow architecture. To balance resource usage and throughput, we apply folding: a time-multiplexing technique that reuses compute units across neurons or channels, reducing hardware cost at the expense of increased inference latency. FINN allows runtime weight updates without resynthesis, which enables weight modification on deployed hardware. All accelerators are synthesized for a 50 MHz clock and support runtime-rewritable weights.

6.2.4.3. Fault Injection

To evaluate our detection method, we conduct two types of fault injection: faults in weights and SAF. Since FINN does not natively support partial or targeted weight updates, we implement a custom function for bit-flips that selectively modifies weight files and writes them back into a format readable for the FINN accelerator. We evaluate fault intensities from 0% to 10%, applied uniformly at random.

For SAFs, we insert dedicated fault injection IP blocks between accelerator layers, without modifying the remaining hardware or introducing latency. The impact of such faults is amplified by folding, as hardware is reused across computations single faults may affect multiple neurons.

6.2.4.4. Measurements

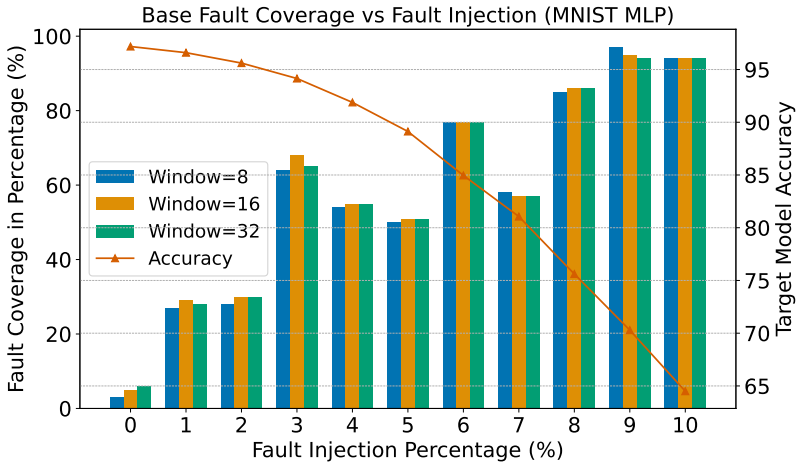
For each fault intensity, we collect 100,000 traces at 100 MHz for both training and evaluation. The sampling rate is chosen to be twice the highest frequency component we aim to capture. Since we target 50 MHz, we sample at 100 MHz to avoid aliasing. The duration of trace collection varies by model, as deeper networks require more inference cycles.

To evaluate detection performance, we construct balanced datasets with a mixed-fault-percentage scheme. The training set includes 80k clean and 8k traces for each of 10 fault intensities. Validation consists of 10k clean and 1k faulty samples per level. The test set contains 10k clean and 10k traces for each fault intensity.

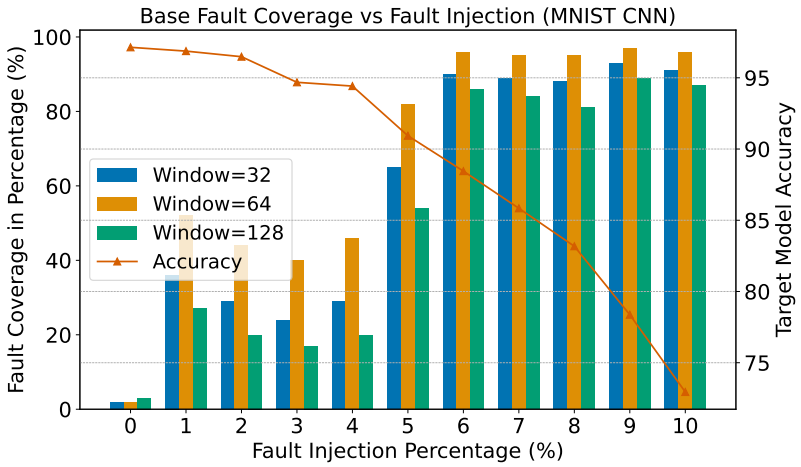
6.2.5. Results

6.2.5.1. Bit-Flips in Memory

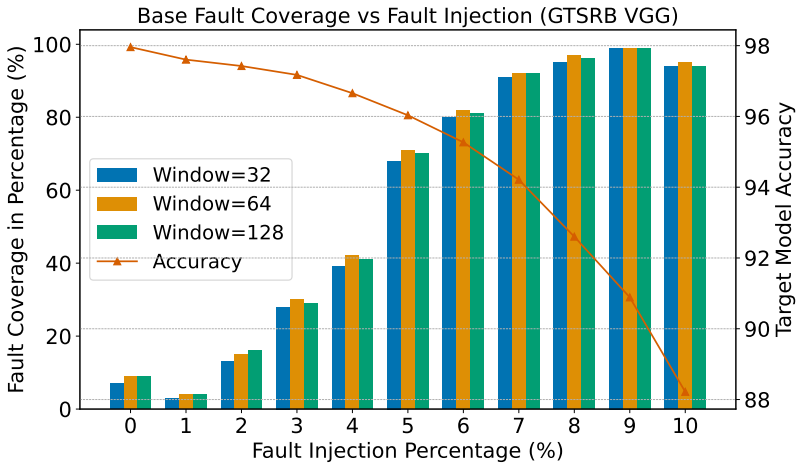
We first assess the base detection performance of our method for single inferences under bit-flip faults. As illustrated in Figure 6.13, the *fault coverage*, i.e., the percentage of correctly identified faulty instances, increases with fault intensity. The accompanying



(a) MNIST MLP



(b) MNIST CNN



(c) GTSRB VGG

Figure 6.13.: Base fault coverage of the detector without detection over time. The target model accuracy with varying fault intensity shows the coverage depending on the inflicted accuracy degradation. From this analysis the optimal window size can be selected.

Table 6.6.: Required Time for Weight Bit Flip Detection (in seconds) across varying FPS and Fault Intensities

Model	W	30 FPS			60 FPS			Max Throughput		
		1%	5%	10%	1%	5%	10%	1%	5%	10%
MNIST MLP	8	1.43	0.57	0.10	0.72	0.28	0.05	2.39×10^{-4}	9.44×10^{-5}	1.67×10^{-5}
MNIST CNN	64	1.00	0.33	0.23	0.50	0.17	0.12	3.86×10^{-3}	1.29×10^{-3}	9.00×10^{-4}
GTSRB VGG	64	–	0.37	0.17	–	0.18	0.08	–	1.81×10^{-3}	8.22×10^{-4}

model accuracy curves show the accuracy degradation at each fault level, providing context for the severity of faults that our method is able to capture. As higher fault intensity leads to larger deviations in voltage fluctuation, faults become easier to detect. At 0% injected faults (fault-free operation) we report very few false positives. This analysis helps to identify a window size that balances coverage and false positive rate, e.g. W=64 for the CNN.

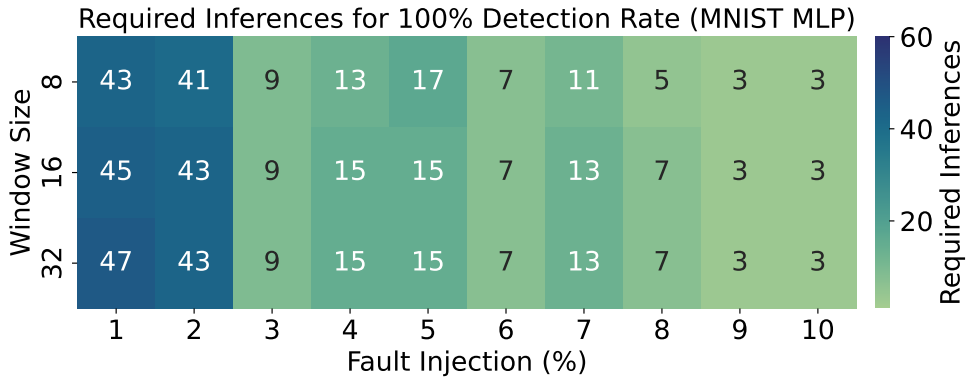
When applying detection over time, we can consistently reach above 99% fault coverage without false positives. Figure 6.14 shows the required number of inferences across window sizes and fault intensities. As expected, higher fault intensities require fewer inferences for reliable detection. CNNs tend to require fewer inferences than the MLP, likely due to the weight sharing in convolutional layers, which amplifies the effect of bit flips. For the GTSRB model faults with 1% intensity could not be detected within acceptable time, however, the accuracy drop in this case is also negligible ($< 0.5\%$).

6.2.5.2. Stuck-At Faults

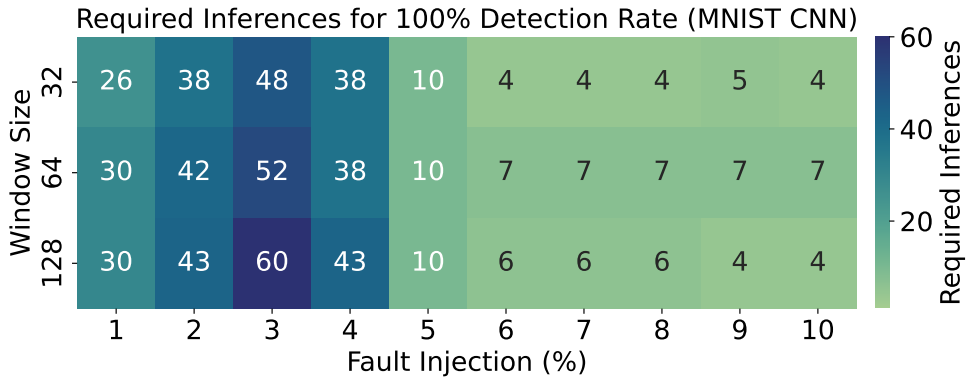
Figure 6.15 shows the number of inferences required to reliably detect SAF with above 99% coverage and no false positives for both the MNIST MLP and CNN models. For each model, we evaluate different window sizes and fault injection rates from 1% to 10%. In the MLP (Figure 6.15a), lower fault intensities such as 1–3% require up to 25 inferences for detection, whereas higher intensities are detected with as few as 15 inferences. In contrast, the CNN model (Figure 6.15b) allows more reliable detection of SAFs, consistently requiring only 3–5 inferences across all intensities and window sizes. We omit GTSRB, as for this model SAFs could be detected with 100% accuracy with single inferences. Here, detection over time is used only to eliminate faults positives (3 inferences).

6.2.5.3. Hardware Implementation

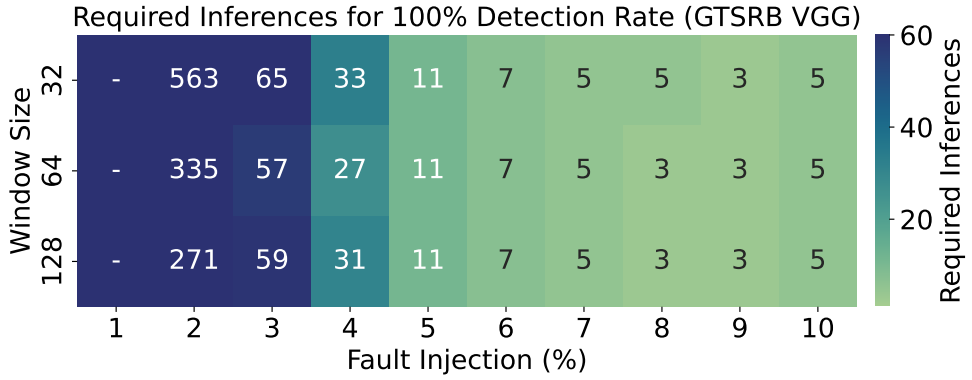
The full detection pipeline is implemented on the FPGA. Figure 6.16 shows the floorplan of a deployed CNN accelerator including four RDS and the detector. The CNN occupies the majority of logic (highlighted in yellow), while four 255-bit RDS blocks are shown in purple. The fault detection module, highlighted in blue in the bottom right corner occupies only a small portion of the FPGA.



(a) MNIST MLP



(b) MNIST CNN

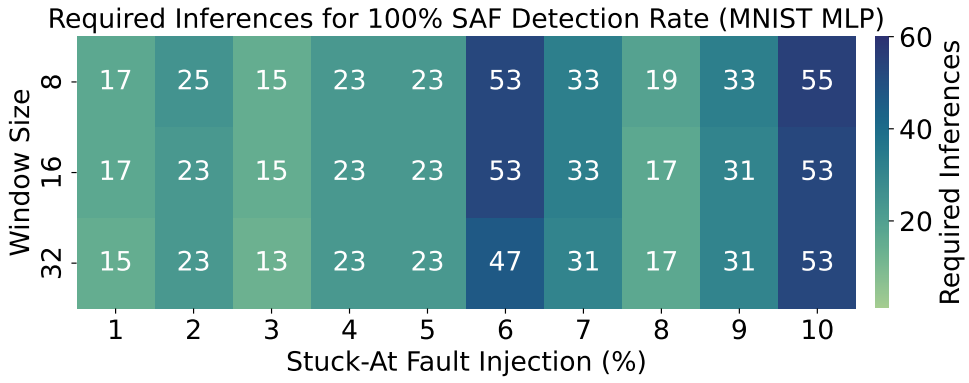


(c) GTSRB VGG

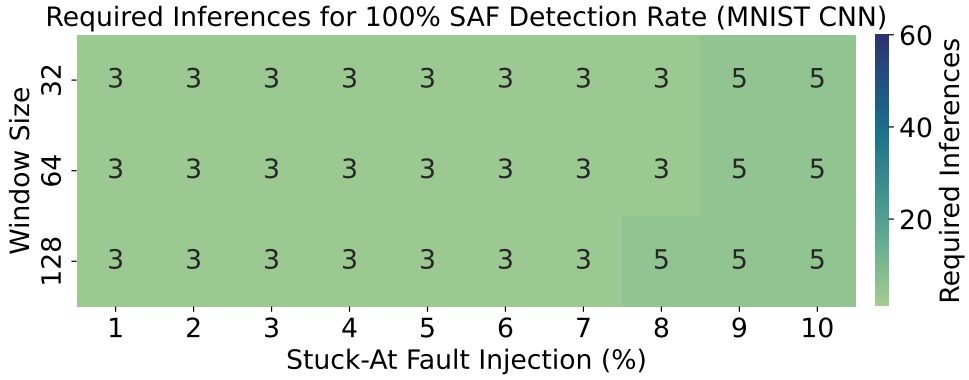
Figure 6.14.: Required #inferences for >99% weight bit-flip detection.

Table 6.6 shows the time required to reliably detect weight bit-flip faults at different fault intensities and frame rates. For applications such as vision systems running at 30–60 FPS, our method achieves sub-second detection times across most tested scenarios. Under high-throughput operation (maximum throughput of the accelerator), detection is even possible in under 1 millisecond, enabling extremely fast response.

We analyze the hardware overhead of the detector as reported by Vivado for the PYNQ-Z2. As shown in Table 6.7, our detection logic consumes a negligible fraction of the FPGA



(a) MNIST MLP



(b) MNIST CNN

Figure 6.15.: Required #inferences for >99% logic fault detection. GTSRB is omitted, as it achieves 100% accuracy for any fault intensity.

Component	LUTs (53200)	Registers (106400)	BRAMs (140)
MNIST MLP	27046 (50.84%)	45817 (86.12%)	1.5 (1.07%)
Detector (W=8)	61 (0.11%)	69 (0.06%)	0.5 (0.36%)
MNIST CNN	9868 (18.55%)	13486 (12.67%)	16 (11.43%)
Detector (W=64)	55 (0.10%)	81 (0.08%)	1 (0.71%)
RDS (4 Sensors)	2075 (3.90%)	1925 (1.81%)	0

Table 6.7.: FPGA Resource Utilization on PYNQ-Z2

resources. For instance, in the MNIST CNN case, the detector uses less than 0.1% of LUTs and registers, and only a single BRAM block. The RDS modules, which enable fine-grained voltage sampling, account for 3.9% of LUTs and 1.8% of registers—despite using four parallel sensors to accelerate our experiments. For deployment, a single sensor is sufficient, which would further reduce the already modest resource usage.

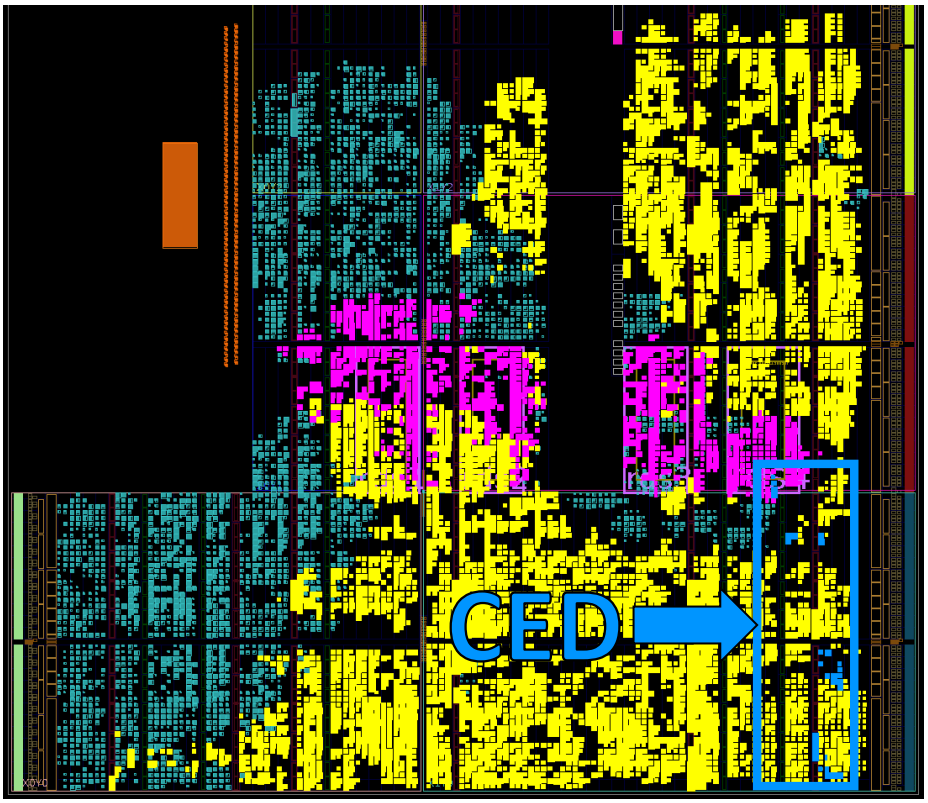


Figure 6.16.: Floorplan showing CNN (yellow), RDS sensors (pink), and our CED (blue, bottom right, highlighed by box).

6.2.6. Section Discussion

Our evaluation demonstrates that the proposed detector can be deployed in continuous inference settings without compromising responsiveness or throughput. As shown in Table 6.6, our method consistently achieves sub-second detection across most tested fault intensities and models, and detection can be triggered in a few milliseconds at maximum throughput. Furthermore, by configuring the threshold, the sensitivity of the detector can be adjusted. This allows running the system with pre-determined tolerance to faults.

Notably, our detection model was trained exclusively on voltage traces resulting from weight bit flips, yet it generalizes effectively to SAFs as well. This suggests that both fault types introduce sufficiently distinct perturbations in the voltage trace to be captured by our detector. We observe a strong detection performance for SAFs in CNN architectures compared to the MLP. This is likely due to the nature of convolutional layers: a single SAF affects multiple output activations as the same kernel slides across an image. Weight sharing increases the spatial influence of faults, leading to more pronounced and repeatable voltage fluctuations.

Interestingly, we also find that detection can become more challenging as the SAF intensity increases. While this may seem counterintuitive, it can be explained by the increased variability in trace patterns when many faults are simultaneously active. At low fault rates,

traces exhibit clean deviations from the baseline and as more faults are injected, these deviations may interfere with each other.

We present a lightweight fault detection method for FPGA-based NN accelerators using on-chip voltage fluctuation traces. Our approach enables concurrent detection without requiring reference inputs, redundant execution, or modifications to the inference pipeline. Our experiments show that sub-second detection is reliably achieved in continuous inference systems running at 30–60 FPS. Notably, although the model was only trained on weight faults, it generalizes well to SAF, particularly in CNNs, where shared weights and hardware reuse amplify fault effects. The method remains effective even under varying fault intensities, though high SAF levels can introduce increased trace variability, slightly reducing separability. Overall, our results demonstrate the deployability of voltage-based detection across both MLP and CNN architectures under diverse fault conditions and heterogeneous FPGA platforms.

7. Conclusion and Future Outlook

7.1. Conclusion

This thesis presented a systematic investigation of side-channel leakage in FPGA-based neural network accelerators, with a particular focus on realistic deployment scenarios and resource-constrained implementations. In contrast to prior work that often relies on proof-of-concept implementations, this thesis showed how architectural optimizations such as folding, pipelining, and quantization directly shape leakage characteristics and attack feasibility in realistic accelerators.

Across multiple contributions, the results demonstrate that side-channel information constitutes a powerful and versatile source of information that can be exploited in different ways. On the offensive side, we showed that even realistic accelerator designs leak sufficient information to recover critical properties such as model architecture, folding parameters, and private inputs. In particular, folding recovery enables accurate reconstruction of network topology and removes an important limitation of earlier reverse-engineering attacks. Furthermore, generative models were shown to reconstruct image and audio inputs from power traces across varying environmental and implementation conditions, underlining the robustness of these attacks.

Beyond passive observation, this thesis also demonstrated that side-channel leakage can be actively amplified during training. By embedding hardware-aware objectives into the optimization process, the correlation between power consumption and model outputs can be significantly increased, enabling output recovery without hardware modifications. These results reveal that leakage is not merely an implementation byproduct, but can also be intentionally shaped during model development.

At the same time, this work explored defensive strategies and showed that leakage can be reduced through side-channel-aware training and exploited for model fingerprinting. While the proposed countermeasures significantly lower attack success, they do not eliminate leakage entirely and should therefore be understood as part of a broader security strategy. This underlines a central challenge for FPGA-based AI systems: improving protection without incurring prohibitive hardware or performance overhead.

Finally, a key contribution of this thesis is the demonstration that side-channel signals are not solely a security liability but can also be leveraged for functional safety. We showed that power traces can support concurrent out-of-distribution detection and fault detection during inference, without modifying the accelerator or introducing latency. These mechanisms operate with minimal hardware overhead and generalize across architectures

and fault types, illustrating the practical viability of side-channel-based monitoring in real-world systems.

Overall, the findings of this thesis establish a dual role of side-channel information in neural network accelerators. On the one hand, it exposes severe confidentiality risks by enabling model theft, input reconstruction, and output leakage. On the other hand, the same physical signals can be repurposed to enhance system robustness and reliability. This duality suggests that side-channel awareness should be treated not only as a security concern, but as a fundamental design consideration for future reconfigurable AI systems.

7.2. Future Outlook

Building on the results of this thesis, several concrete directions emerge for advancing the security and reliability of FPGA-based neural network accelerators. A central next step is the development of more precise and generalizable leakage models that capture the effects of architectural features such as folding, pipelining, quantization, and memory access patterns. Such models would improve the understanding of how data and implementation choices shape physical leakage and could further strengthen both attack evaluation and countermeasure design, for example through hardware-in-the-loop or leakage-aware training.

From a security perspective, designing efficient protection mechanisms that remain compatible with resource-constrained FPGA deployments remains highly relevant. As demonstrated in this work, training-based approaches can reduce leakage without additional hardware overhead, but stronger protection will likely require combining them with complementary techniques such as execution randomization or lightweight hiding mechanisms.

On the functional safety side, side-channel-based monitoring opens up new opportunities for runtime assurance in AI hardware. Extending these methods to larger models, broader fault classes, and more dynamic deployment conditions will be important for improving robustness and validating their use in practical safety-critical systems.

Finally, transferring the presented concepts to emerging accelerator paradigms, such as spiking neural networks, is a promising direction for understanding how event-driven computation and stateful processing influence side-channel behavior, attack surfaces, and monitoring opportunities.

Bibliography

[1] V. Meyers, D. Gnad, and M. Tahoori, "Active and passive physical attacks on neural network accelerators", *IEEE Design & Test*, 2023.

[2] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Leveraging neural trojan side-channels for output exfiltration", *Cryptography*, vol. 9, no. 1, p. 5, 2025.

[3] V. Meyers, D. Gnad, and M. Tahoori, "Reverse engineering neural network folding with remote fpga power analysis", in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2022, pp. 1–10.

[4] L. Huegle, M. Gotthard, V. Meyers, J. Krautter, D. R. Gnad, and M. B. Tahoori, "Power2picture: Using generative cnns for input recovery of neural network accelerators through power side-channels on fpgas", in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2023, pp. 155–161.

[5] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Remote identification of neural network fpga accelerators by power fingerprints", in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2023, pp. 259–264.

[6] D. Gnad, J. Krautter, A. Kritikakou, V. Meyers, P. Rech, J. E. R. Condia, A. Ruospo, E. Sanchez, F. F. dos Santos, O. Sentieys, et al., "Reliability and security of ai hardware", in *ETS 2024-29th IEEE European Test Symposium*, IEEE, 2024, pp. 1–10.

[7] V. Meyers, M. Hefenbrock, D. Gnad, and M. Tahoori, "Trained to leak: Hiding trojan side-channels in neural network weights", in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2024, pp. 122–127.

[8] V. Meyers, D. Gnad, and M. Tahoori, "Out-of-distribution detection using power-side channels for improving functional safety of neural network fpga accelerators", in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2024, pp. 1–2.

[9] V. Meyers, M. Hefenbrock, M. Sadeghipourrudsari, D. Gnad, and M. Tahoori, "Towards functional safety of neural network hardware accelerators: Concurrent out-of-distribution detection in hardware using power side-channel analysis", in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1413–1419.

[10] M. S. Akram, V. Meyers, M. Tahoori, B. S. Varma, and D. Finlay, "Evoweight: Sponge poisoning of fpga-based dnn accelerators in differential private secure federated learning", in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2025, pp. 182–193.

[11] V. Meyers, M. Sadeghipourrudsari, and M. Tahoori, "Concurrent fault detection for binary neural network accelerators via on-chip voltage monitoring", in *2026 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2026.

[12] J. Reibold, V. Meyers, and M. Tahoori, "Talking traces: Audio reconstruction power side-channel attack on neural network fpga accelerators", in *2026 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2026, accepted.

[13] V. Meyers, D. R. Gnad, N. M. Dang, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Stealthy logic misuse for power analysis attacks in multi-tenant fpgas (extended version)", *Cryptology ePrint Archive*, 2023.

[14] M. S. Roodsari, J. Krautter, V. Meyers, and M. Tahoori, "E 3 hdc: Energy efficient encoding for hyper-dimensional computing on edge devices", in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2024, pp. 274–280.

[15] M. S. Akram, B. S. Varma, V. Meyers, M. Tahoori, and D. Finlay, "F2opt: Novel fine-tuning and folding algorithms for fpga-based dnn accelerators", in *2025 35th International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2025.

[16] M. S. Roodsari, V. Meyers, and M. Tahoori, "Ced-hdc: Lightweight concurrent error detection for reliable hyperdimensional computing", in *2025 IEEE 43rd VLSI Test Symposium (VTS)*, IEEE, 2025, pp. 1–7.

[17] M. S. Roodsari, V. Meyers, and M. Tahoori, "Lightweight concurrent out-of-distribution detection in hyperdimensional computing hardware", in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2025, pp. 1–7.

[18] M. Tahoori, V. Meyers, M. Sadeghipour Roodsari, H. Xu, J. Becker, T. Harbaum, F. Frombach, J. Hoefer, G. Sotiropoulos, J. Henkel, et al., "Special sessions-hardware-software co-design for machine learning systems made open-source", in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2025, pp. 23–32.

[19] J. Henkel, M. Tahoori, H. Khdr, H. Nassar, V. Meyers, D. Chen, S. Yildirim, Y. Huang, N. R. Saxena, S. Hukerikar, et al., "Hardware-software co-design for highly optimized, customized, and reliable ai systems", in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2025, pp. 1–9.

[20] H. Xu, S. Meschkov, V. Meyers, and M. Tahoori, "Pwnn: Power-wasting neural network as remote fault injector", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2026, no. 1, pp. 448–471, 2026.

[21] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[22] M. Zhu, L. Liu, C. Wang, and Y. Xie, "Cnnlab: A novel parallel framework for neural networks using gpu and fpga-a practical study with trade-off analysis", *arXiv preprint arXiv:1606.06234*, 2016.

[23] Microsoft Azure. “Azure machine learning - ML as a service”. [Online]. Available: https://azure.microsoft.com/en-us/products/machine-learning.

[24] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, “The em side—channel (s)”, in *International workshop on cryptographic hardware and embedded systems*, Springer, 2002, pp. 29–45.

[25] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis”, in *Annual international cryptology conference*, Springer, 1999, pp. 388–397.

[26] K. M. Zick, M. Srivastav, W. Zhang, and M. French, “Sensing nanosecond-scale voltage attacks and natural transients in FPGAs”, in *FPGA*, 2013, pp. 101–104.

[27] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, “An inside job: Remote power analysis attacks on fpgas”, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1111–1116. DOI: 10.23919/DATE.2018.8342177.

[28] L. Batina, S. Bhasin, D. Jap, and S. Picek, “Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel”, in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC’19, Santa Clara, CA, USA: USENIX Association, 2019, pp. 515–532, ISBN: 9781939133069.

[29] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, “Power side-channel attacks on bnn accelerators in remote fpgas”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 357–370, 2021. DOI: 10.1109/JETCAS.2021.3074608.

[30] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, “Stealing neural network structure through remote fpga side-channel analysis”, *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021. DOI: 10.1109/TIFS.2021.3106169.

[31] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, “Finn: A framework for fast, scalable binarized neural network inference”, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’17, ACM, 2017, pp. 65–74.

[32] ONNX Community, *Open neural network exchange (onnx)*, Accessed: 2024-06-12: https://onnx.ai, 2024.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library”, in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[34] G. Franco, A. Pappalardo, and N. J. Fraser, *Xilinx/brevitas*, 2025. DOI: 10.5281/zenodo.3333552. [Online]. Available: https://doi.org/10.5281/zenodo.3333552.

[35] A. Pappalardo, Y. Umuroglu, M. Blott, J. Mitrevski, B. Hawks, N. Tran, V. Loncar, S. Summers, H. Borras, J. Muhizi, M. Trahms, S.-C. H. Hsu, S. Hauck, and J. Duarte, "QONNX: Representing Arbitrary-Precision Quantized Neural Networks", in *4th Workshop on Accelerated Machine Learning (AccML) at HiPEAC 2022 Conference*, Jun. 2022. arXiv: 2206.07527 [cs.LG].

[36] S. I. Venieris and C.-S. Bouganis, "Fpgaconvnet: A framework for mapping convolutional neural networks on fpgas", in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2016, pp. 40–47.

[37] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, "A gpu-outperforming fpga accelerator architecture for binary convolutional neural networks", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, pp. 1–16, 2018.

[38] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas", in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, Monterey, California, USA: Association for Computing Machinery, 2017, pp. 15–24, ISBN: 9781450343541. DOI: 10.1145/3020078.3021741.

[39] T. Moreau, T. Chen, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Vta: An open hardware-software stack for deep learning", *arXiv preprint arXiv:1807.04188*, 2018.

[40] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud fpgas really vulnerable to power analysis attacks?", in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 1007–1010.

[41] A. Le Masle and W. Luk, "Detecting power attacks on reconfigurable hardware", in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2012, pp. 14–19.

[42] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks", in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 229–244.

[43] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant fpgas in the cloud", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–26, 2019.

[44] D. R. Gnad, J. Krautter, and M. B. Tahoori, "Remote physical attacks on fpgas at the electrical level", in *Security of FPGA-Accelerated Cloud Computing Environments*, Springer, 2023, pp. 81–99.

[45] D. Spielmann, O. Glamočanin, and M. Stojilović, "Rds: Fpga routing delay sensors for effective remote power analysis attacks", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, pp. 543–567, Mar. 2023. DOI: 10.46586/tches.v2023.i2.543-567.

[46] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection", in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 197–208. DOI: 10.1109/HOST45689.2020.9300276.

[47] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, et al., "A configurable cloud-scale dnn processor for real-time ai", in *International Symposium on Computer Architecture (ISCA)*, ACM/IEEE, 2018, pp. 1–14.

[48] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Fpgahammer: Remote voltage fault attacks on shared fpgas, suitable for dfa on aes", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[49] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators", in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18, San Juan, PR, USA: Association for Computing Machinery, 2018, pp. 393–406, ISBN: 9781450365697. DOI: 10.1145/3274694.3274696.

[50] W. Liu, C.-H. Chang, F. Zhang, and X. Lou, "Imperceptible misclassification attack on deep learning accelerator by glitch injection", in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[51] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote power attacks on the versatile tensor accelerator in multi-tenant fpgas", in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2021, pp. 242–246.

[52] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through em side-channel information leakage", in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 209–218. DOI: 10.1109/HOST45689.2020.9300274.

[53] J. Breier, D. Jap, X. Hou, S. Bhasin, and Y. Liu, "Sniff: Reverse engineering of neural networks with fault attacks", *Transactions on Reliability*, 2021.

[54] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in Multi-Tenant FPGA", in *30th USENIX Security Symposium (USENIX Security)*, USENIX Association, Aug. 2021, pp. 1919–1936, ISBN: 978-1-939133-24-3.

[55] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{Csi}{nn}: Reverse engineering of neural network architectures through electromagnetic side channel", in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 515–532.

[56] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels–attacks and defenses", *IEEE Internet of Things Journal*, 2021.

[57] G. Takatoi, T. Sugawara, K. Sakiyama, and Y. Li, "Simple electromagnetic analysis against activation functions of deep neural networks", in *International Conference on Applied Cryptography and Network Security*, Springer, 2020, pp. 181–197.

[58] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang, "Open dnn box by power side-channel attack", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 11, pp. 2717–2721, 2020.

[59] V. Yli-Mäyry, A. Ito, N. Homma, S. Bhasin, and D. Jap, "Extraction of binarized neural network architecture and secret parameters using side-channel information", in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[60] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories", in *Symposium on Security and Privacy (S&P)*, IEEE, 2022, pp. 1157–1174.

[61] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier, "Remote power side-channel attacks on cnn accelerators in fpgas", *arXiv preprint arXiv:2011.07603*, 2020.

[62] L. Wu, L. Wu, Z. Ba, and X. Zhang, "An input recovery side-channel attack on dnn accelerator with three-dimensional power surface", in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2025, pp. 1–11. DOI: 10.1109/HOST64725.2025.11050042.

[63] G. Dong, P. Wang, P. Chen, R. Gu, and H. Hu, "Floating-point multiplication timing attack on deep neural network", in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2019, pp. 155–161. DOI: 10.1109/SmartIoT.2019.00032.

[64] Z. Wang, Y. Wu, Y. Park, S. Yoo, X. Wang, J. K. Eshraghian, and W. D. Lu, "Powergan: A machine learning approach for power side-channel attack on compute-in-memory accelerators", *Advanced Intelligent Systems*, vol. 5, no. 12, p. 2 300 313, 2023. DOI: https://doi.org/10.1002/aisy.202300313. eprint: https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202300313. [Online]. Available: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202300313.

[65] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks", *arXiv preprint arXiv:1610.02136*, 2016.

[66] M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem", in *Conference on Computer Vision and Pattern Recognition*, IEEE, 2019.

[67] R. Chan, M. Rottmann, and H. Gottschalk, "Entropy maximization and meta classification for out-of-distribution detection in semantic segmentation", in *International Conference on Computer Vision*, IEEE, 2021.

[68] W. Liu, X. Wang, J. Owens, and Y. Li, "Energy-based out-of-distribution detection", *Advances in Neural Information Processing Systems*, 2020.

[69] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks", *arXiv preprint arXiv:1706.02690*, 2017.

[70] A. Neale and M. Sachdev, "Neutron radiation induced soft error rates for an adjacent-ecc protected sram in 28 nm cmos", *IEEE Transactions on Nuclear Science*, vol. 63, no. 3, pp. 1912–1917, 2016.

[71] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors", *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1215–1228, 2012.

[72] P. Koopman and T. Chakravarty, "Cyclic redundancy code (crc) polynomial selection for embedded networks", in *International Conference on Dependable Systems and Networks, 2004*, IEEE, 2004, pp. 145–154.

[73] Y. Li, M. Li, B. Luo, Y. Tian, and Q. Xu, "Deepdyve: Dynamic verification for deep neural networks", in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 101–112.

[74] F. Geissler, S. Qutub, S. Roychowdhury, A. Asgari, Y. Peng, A. Dhamasia, R. Graefe, K. Pattabiraman, and M. Paulitsch, "Towards a safety case for hardware fault tolerance in convolutional neural networks using activation range supervision", *arXiv preprint arXiv:2108.07019*, 2021.

[75] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, "Radar: Run-time adversarial weight attack detection and accuracy recovery", in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2021, pp. 790–795.

[76] M. Javaheripi and F. Koushanfar, "Hashtag: Hash signatures for online detection of fault-injection attacks on deep neural networks", in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2021, pp. 1–9.

[77] P. N. Fahn and P. K. Pearson, "Ipa: A new class of power attacks", in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer, 1999, pp. 173–186.

[78] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks", in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer, 2002, pp. 13–28.

[79] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)", *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[80] T. K. Ho, "Random decision forests", in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, 1995, pp. 278–282.

[81] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2007, ISBN: 978-0-387-31073-2.

[82] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost", *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[83] J. H. Friedman, "Stochastic gradient boosting", *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[84] A. Heuser, M. Kasper, W. Schindler, and M. Stöttinger, "A new difference method for side-channel analysis with high-dimensional leakage models", in *Cryptographers' Track at the RSA Conference*, Springer, 2012, pp. 365–382.

[85] A. Dubey, R. Cammarota, and A. Aysu, "Bomanet: Boolean masking of an entire neural network", in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380263. DOI: 10.1145/3400302.3415649.

[86] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active fences against voltage-based side channels in multi-tenant fpgas", in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2019, pp. 1–8.

[87] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks", in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018, pp. 1–6.

[88] A. Rajkomar, J. Dean, and I. Kohane, "Machine learning in medicine", *New England Journal of Medicine*, vol. 380, no. 14, pp. 1347–1358, 2019. DOI: 10.1056/NEJMra1814259.

[89] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks", *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020, ISSN: 0001-0782. DOI: 10.1145/3422622. [Online]. Available: https://doi.org/10.1145/3422622.

[90] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[91] H. Xiao, K. Rasul, and R. Vollgraf. "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms". arXiv: cs.LG/1708.07747 [cs.LG].

[92] TensowFlow Core. "Deep convolutional generative adversarial network". [Online]. Available: https://www.tensorflow.org/tutorials/generative/dcgan.

[93] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[94] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity", *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861.

[95] Y. LeCun, "The mnist database of handwritten digits", *http://yann.lecun.com/exdb/mnist/*, 1998.

[96] Weisstechnik. "EMC test chambers LabEvent". [Online]. Available: https://backend.weiss-technik.com/webapp/weisstechnik/multimedia-center/brochures/2021/Weiss-Technik-LabEvent-EMV-EN-1.pdf.

[97] H. Guntur, J. Ishii, and A. Satoh, "Side-channel attack user reference architecture board SAKURA-G", in *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, IEEE, 2014, pp. 271–274.

[98] A. Natsiou and S. O'Leary, "Audio representations for deep learning in sound synthesis: A review", *CoRR*, vol. abs/2201.02490, 2022. arXiv: 2201.02490.

[99] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, "Deep learning for audio signal processing", *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019. DOI: 10.1109/JSTSP.2019.2908700.

[100] K. Kumar, R. Kumar, T. De Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. De Brebisson, Y. Bengio, and A. C. Courville, "Melgan: Generative adversarial networks for conditional waveform synthesis", *Advances in neural information processing systems*, vol. 32, 2019.

[101] J. Kong, J. Kim, and J. Bae, "Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 17 022–17 033.

[102] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music", in *International conference on machine learning*, PMLR, 2018, pp. 4364–4373.

[103] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis", in *ICLR*, 2019.

[104] G. Zhu, Y. Wen, M.-A. Carbonneau, and Z. Duan, "Edmsound: Spectrogram based diffusion models for efficient and high-quality audio synthesis", *arXiv preprint arXiv:2311.08667*, 2023.

[105] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984. DOI: 10.1109/TASSP.1984.1164317.

[106] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures", Aug. 2017, pp. 45–68, ISBN: 978-3-319-66786-7. DOI: 10.1007/978-3-319-66787-4_3.

[107] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, "Very deep convolutional neural networks for raw waveforms", in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 421–425. DOI: 10.1109/ICASSP.2017.7952190.

[108] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[109] M. R. Saddler, A. Francl, J. Feather, K. Qian, Y. Zhang, and J. H. McDermott, "Speech denoising with auditory models", *arXiv preprint arXiv:2011.10706*, 2020.

[110] C. J. Steinmetz and J. D. Reiss, "Auraloss: Audio focused loss functions in PyTorch", in *Digital Music Research Network One-day Workshop (DMRN+15)*, 2020.

[111] Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment", in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2, 2003, 1398–1402 Vol.2. DOI: 10.1109/ACSSC.2003.1292216.

[112] T. Namgyal, A. Hepburn, R. Santos-Rodriguez, V. Laparra, and J. Malo, *What you hear is what you see: Audio quality metrics from image quality metrics*, 2023. arXiv: 2305.11582 [cs.SD].

[113] G. Fang, *Pytorch ms-ssim*. Accessed: Jun. 18, 2025. [Online]. Available: https://github.com/VainF/pytorch-msssim.

[114] C. Shannon, "Communication in the presence of noise", *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949. DOI: 10.1109/JRPROC.1949.232969.

[115] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition", *CoRR*, vol. abs/1804.03209, 2018. arXiv: 1804.03209. [Online]. Available: http://arxiv.org/abs/1804.03209.

[116] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and music signal analysis in python.", *SciPy*, vol. 2015, pp. 18–24, 2015.

[117] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[118] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using fpga-accelerated neural networks", *BMC bioinformatics*, vol. 19, pp. 19–31, 2018.

[119] P. Jokic, S. Emery, and L. Benini, "Binaryeye: A 20 kfps streaming camera system on fpga with real-time on-device image recognition using binary neural networks", in *International Symposium on Industrial Embedded Systems (SIES)*, IEEE, 2018, pp. 1–7.

[120] N. Fasfous, M.-R. Vemparala, A. Frickenstein, L. Frickenstein, M. Badawy, and W. Stechele, "Binarycop: Binary neural network-based covid-19 face-mask wear and positioning predictor on edge devices", in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2021, pp. 108–115.

[121] P. Warden, *Speech commands: A dataset for limited-vocabulary speech recognition*, 2018. arXiv: 1804.03209 [cs.CL].

[122] R. Koppel and C. E. Kuziemsky, "Healthcare data are remarkably vulnerable to hacking: Connected healthcare delivery increases the risks.", in *ITCH*, 2019, pp. 218–222.

[123] A. Downing and E. Perakslis, "Health advertising on facebook: Privacy and policy considerations", *Patterns*, vol. 3, no. 9, 2022.

[124] O. Söll, T. Korak, M. Muehlberghuber, and M. Hutter, "EM-based detection of hardware trojans on fpgas", in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2014, pp. 84–87.

[125] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering", in *CHES*, Springer, 2009, pp. 382–395.

[126] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, 2013. arXiv: 1308.3432 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1308.3432.

[127] A. Dubey, E. Karabulut, A. Awad, and A. Aysu, "High-fidelity model extraction attacks via remote power monitors", in *Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2022, pp. 328–331.

[128] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[129] A. Krizhevsky, G. Hinton, et al., "Learning multiple layers of features from tiny images", 2009.

[130] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification", *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[131] Z. Wang, B. Che, L. Guo, Y. Du, Y. Chen, J. Zhao, and W. He, "Pipefl: Hardware/software co-design of an fpga accelerator for federated learning", *IEEE Access*, vol. 10, pp. 98 649–98 661, 2022.

[132] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, "Federated learning with differential privacy: Algorithms and performance analysis", *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575.

[133] J. Liu, Z. Zhu, Y. Zhou, N. Wang, G. Dai, Q. Liu, J. Xiao, Y. Xie, Z. Zhong, H. Liu, et al., "Bioaip: A reconfigurable biomedical ai processor with adaptive learning for versatile intelligent health monitoring", in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, vol. 64, 2021, pp. 62–64.

[134] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities", in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, 2017, pp. 1–8.

[135] Z. Chen, J. Luo, K. Lin, J. Wu, T. Zhu, X. Xiang, and J. Meng, "An energy-efficient ecg processor with weak-strong hybrid classifier for arrhythmia detection", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 7, pp. 948–952, 2017.

[136] J. Nurmi, Y. Xu, J. Boutellier, and B. Tan, "Sphere-dna: Privacy-preserving federated learning for ehealth", in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[137] M. S. Akram, B. S. Varma, and D. Finlay, "Dpfl-fpga-accel: Open source framework for design space exploration of fpga-based differential private federated learning accelerator: A case study with cardiac arrhythmia", *Authorea Preprints*, 2024.

[138] M. T. Hossain, S. Islam, S. Badsha, and H. Shen, "Desmp: Differential privacy-exploited stealthy model poisoning attacks in federated learning", in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2021, pp. 167–174.

[139] M. B. Tahoori, "Security challenges and opportunities of cloud fpgas", in *Proceedings of the 2023 on Cloud Computing Security Workshop*, 2023, pp. 1–1.

[140] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, "Sponge examples: Energy-latency attacks on neural networks", in *2021 IEEE European symposium on security and privacy (EuroS&P)*, IEEE, 2021, pp. 212–231.

[141] S. Paul and N. Kourtellis, "Sponge ml model attacks of mobile apps", in *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*, 2023, pp. 139–139.

[142] J. t. Lintelo, S. Koffas, and S. Picek, "The skipsponge attack: Sponge weight poisoning of deep neural networks", *arXiv preprint arXiv:2402.06357*, 2024.

[143] J. Zhang, H. Zhu, F. Wang, J. Zhao, Q. Xu, and H. Li, "Security and privacy threats to federated learning: Issues, methods, and challenges", *Security and Communication Networks*, vol. 2022, no. 1, p. 2 886 795, 2022.

[144] M. Benmalek, M. A. Benrekia, and Y. Challal, "Security of federated learning: Attacks, defensive mechanisms, and challenges", *Revue des Sciences et Technologies de l'Information-Série RIA: Revue d'Intelligence Artificielle*, vol. 36, no. 1, pp. 49–59, 2022.

[145] T. D. Nguyen, P. Rieger, R. De Viti, H. Chen, B. B. Brandenburg, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, et al., "{Flame}: Taming backdoors in federated learning", in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1415–1432.

[146] H. Li, P. Rieger, S. Zeitouni, S. Picek, and A.-R. Sadeghi, "Flairs: Fpga-accelerated inference-resistant & secure federated learning", in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2023, pp. 271–276.

[147] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning", in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 2938–2948.

[148] A. E. Cinà, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "Energy-latency attacks via sponge poisoning", *arXiv preprint arXiv:2203.08147*, 2023.

[149] Z. Zhang, A. Panda, L. Song, Y. Yang, M. Mahoney, P. Mittal, R. Kannan, and J. Gonzalez, "Neurotoxin: Durable backdoors in federated learning", in *International Conference on Machine Learning*, PMLR, 2022, pp. 26 429–26 446.

[150] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference", *arXiv preprint arXiv:2211.05102*, 2022.

[151] M. S. Akram, B. S. Varma, and D. Finlay, "Continual learning on fpgas for efficient cardiac diagnosis through mix-precision quantized dnns", *Authorea Preprints*, 2024.

[152] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[153] Xilinx, *Aup pynq-z2*, Accessed 2024-11-05: `https://www.amd.com/de/corporate/university-program/aup-boards/pynq-z2`, 2024.

[154] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, ..., and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals", *Circulation [Online]*, vol. 101, no. 23, e215–e220, 2000.

[155] M. S. Akram, B. S. Varma, and D. Finlay, "Embedded dnn classifier for five different cardiac diseases", in *2024 35th Irish Signals and Systems Conference (ISSC)*, IEEE, 2024, pp. 01–06.

[156] Heart Rhythm Society, *Heart rhythm disorders*, Available at: `https://upbeat.org/heart-rhythm-disorders`, 2022.

[157] Xilinx, *Pmbus*, Accessed 2024-11-05: `https://github.com/Xilinx/PYNQ/blob/image_v2.4/pynq/pmbus.py`, 2024.

[158] Teledyne LeCroy, *Hdo6104a high definition oscilloscope*, Accessed 2024-11-05: `https://www.teledynelecroy.com/oscilloscope`, 2024.

[159] X. Cao, J. Jia, and N. Z. Gong, "IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary", in *ACM Asia Conference on Computer and Communications Security*, 2021, pp. 14–25.

[160] Z. He, T. Zhang, and R. Lee, "Sensitive-sample fingerprinting of deep neural networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4729–4737.

[161] T. Dong, H. Qiu, T. Zhang, J. Li, H. Li, and J. Lu, "Fingerprinting multi-exit deep neural network models via inference time", *arXiv preprint arXiv:2110.03175*, 2021.

[162] N. Lukas, Y. Zhang, and F. Kerschbaum, "Deep neural network fingerprinting by conferrable adversarial examples", *arXiv preprint arXiv:1912.00888*, 2019.

[163] S. Wang and C.-H. Chang, "Fingerprinting deep neural networks-a deepfool approach", in *International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[164] S. Wang, P. Zhao, X. Wang, S. Chin, T. Wahl, Y. Fei, Q. A. Chen, and X. Lin, "Intrinsic examples: Robust fingerprinting of deep neural networks", in *British Machine Vision Conference (BMVC)*, 2021.

[165] F. Ding, J.-S. Denain, and J. Steinhardt, "Grounding representation similarity through statistical testing", *Advances in Neural Information Processing Systems*, vol. 34, pp. 1556–1568, 2021.

[166] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al., "A survey of uncertainty in deep neural networks", *arXiv preprint arXiv:2107.03342*, 2021.

[167] O. Glamocanin, H. Bazaz, M. Payer, and M. Stojilovic, "Temperature impact on remote power side-channel attacks on shared fpgas", in *Design, Automation and Test in Europe Conference (DATE)*, 2023.

[168] T. DeVries and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks", *arXiv preprint arXiv:1802.04865*, 2018.

[169] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning", in *International Conference on Machine Learning*, PMLR, 2016.

[170] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark", in *International Joint Conference on Neural Networks*, 2013.

[171] N. Mu and J. Gilmer, "Mnist-c: A robustness benchmark for computer vision", *arXiv preprint arXiv:1906.02337*, 2019.

[172] M. A. mnmoustafa, *Tiny imagenet*, 2017. [Online]. Available: `https://kaggle.com/competitions/tiny-imagenet`.

[173] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al., "Reading digits in natural images with unsupervised feature learning", in *NIPS workshop on deep learning and unsupervised feature learning*, Granada, vol. 2011, 2011, p. 4.

[174] K. Kirchheim, M. Filax, and F. Ortmeier, "Pytorch-ood: A library for out-of-distribution detection based on pytorch", in *International Conference on Computer Vision and Pattern Recognition*, IEEE, 2022.

[175] I. Kouretas and V. Paliouras, "Hardware implementation of a softmax-like function for deep learning", *Technologies*, vol. 8, no. 3, p. 46, 2020.

[176] J. Yang, P. Wang, D. Zou, Z. Zhou, K. Ding, W. Peng, H. Wang, G. Chen, B. Li, Y. Sun, et al., "Openood: Benchmarking generalized out-of-distribution detection", *Advances in Neural Information Processing Systems*, 2022.

[177] D. Hendrycks, S. Basart, M. Mazeika, A. Zou, J. Kwon, M. Mostajabi, J. Steinhardt, and D. Song, "Scaling out-of-distribution detection for real-world settings", *arXiv preprint arXiv:1911.11132*, 2019.

[178] D. Xu, Z. Zhu, C. Liu, Y. Wang, S. Zhao, L. Zhang, H. Liang, H. Li, and K.-T. Cheng, "Reliability evaluation and analysis of fpga-based neural network acceleration system", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 472–484, 2021.

[179] I. Catalán, J. Flich, and C. Hernández, "Exploiting neural networks bit-level redundancy to mitigate the impact of faults at inference", *The Journal of Supercomputing*, vol. 81, no. 1, p. 183, 2025.

[180] R. W. Hamming, "Error detecting and error correcting codes", *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[181] W. Li, G. Ge, K. Guo, X. Chen, Q. Wei, Z. Gao, Y. Wang, and H. Yang, "Soft error mitigation for deep convolution neural network on fpga accelerators", in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2020, pp. 1–5.

A. Appendix

List of Figures

List of Tables