

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

Vincent Meyers, Mahboobe Sadeghipour Roodsari, Mehdi Tahoori
Chair of Dependable Nano Computing, Karlsruhe Institute of Technology, Germany
 {vincent.meyers, mahboobe.sadeghipourrudsari, mehdi.tahoori}@kit.edu

Abstract—As Neural Networks (NNs) are increasingly deployed in safety-critical edge and datacenter systems, ensuring reliable execution becomes essential. Runtime faults such as memory bit flips and faults in logic components can silently corrupt computations without triggering system-level alarms. Conventional detection methods often miss logic faults or incur significant overhead. We propose a lightweight, concurrent error detection method that monitors voltage fluctuation traces captured by on-chip sensors. Our hypothesis is that faults alter neuron activations and change the switching activity and thus the instantaneous voltage fluctuation profile during inference. These traces are classified using a threshold-based model, requiring no modifications to the NN hardware or inference pipeline. As our approach operates purely through side-channel observation, it functions as a non-intrusive wrapper applicable to a wide range of AI accelerators. We evaluate the method on two different FPGAs, demonstrating consistent efficiency across platforms and portability to cloud scenarios. It detects faults in under a second, making it suitable for real-time applications such as vision tasks running at 30–60 FPS. By repurposing voltage sensors as diagnostic tools, this work opens a new direction for functional safety in AI hardware.

Index Terms—voltage sensors, NN accelerators, online fault detection

I. INTRODUCTION

Machine learning is now ubiquitous across safety-critical applications in both edge [1] and datacenter infrastructures [2], [3], offering powerful capabilities for real-time decision-making and large-scale data processing. Applications range from autonomous driving [4] to industrial monitoring [5] and medical devices [6]. Edge devices, such as smartphones, enable low-latency inference with enhanced privacy by processing data locally [1]. In contrast, datacenters leverage high-performance accelerators to support massive neural workloads and continuous training, with specialized hardware like Tensor Processing Units (TPUs) and Field Programmable Gate Arrays (FPGAs) delivering high throughput and energy efficiency [3].

To meet these computational demands, hardware accelerators such as Graphics Processing Unit (GPU), TPU, Application Specific Integrated Circuits (ASICs), and FPGAs play a vital role in both edge and datacenter AI deployments [7]. FPGAs stand out due to their reconfigurability, parallelism, and energy efficiency, allowing the design of customized AI accelerators. However, the increasing reliance on specialized hardware in critical applications also introduces new challenges in ensuring fault-tolerant operation [8].

NN accelerators are vulnerable to a wide range of runtime faults in both, the memory and logic components. These may arise from permanent sources such as latent defects or aging, as well as transient effects like radiation-induced soft errors or voltage noise [9]. Even single-bit flips in weights or activations can silently corrupt computations and degrade model accuracy by up to 3% [10]. The complexity of NNs makes fault propagation hard to predict, motivating detection strategies that balance reliability and resource constraints.

Concurrent fault detection in NN accelerators is challenging due to tight performance, area, and power constraints. Several works target this problem using hardware-based solutions [11], [12], [13], but these incur high resource and latency overheads and offer limited coverage for logic faults. Software-level techniques, including duplicate inference [14], activation range checking [15], or hash-based integrity checks [16], improve flexibility but introduce runtime and resource overhead and are tailored to limited fault models. Consequently, there is a pressing need for real-time fault detection methods that operate with minimal system overhead.

In this work, we explore the use of on-chip voltage fluctuation sensors, specifically delay-line-based Routing Delay Sensors (RDS) [17] for detecting faults in NN accelerators. These sensors, commonly used in degradation monitoring [18], [19], temperature sensing [20], and security research [21], exploit the correlation between supply voltage and signal propagation delay to capture subtle voltage variations during execution. Our hypothesis is that hardware faults, such as bit flips or logic errors, alter neuron computations, which affect switching activity and lead to measurable voltage deviations. Prior work has shown this effect can be used for fingerprinting accelerators [22]. By training a lightweight classifier on side-channel information, we enable non-invasive concurrent error detection (CED), requiring no modifications or logical connection to the AI hardware IP.

In summary, the contributions of this paper are as follows:

- Novel fault detection approach leveraging on-chip voltage sensors for monitoring functional faults in NN hardware
- Reliably detects faults affecting both NN weights and logic, addressing a critical gap left by existing fault detection mechanisms
- Enables low-overhead, reference-free fault detection that integrates easily into existing systems

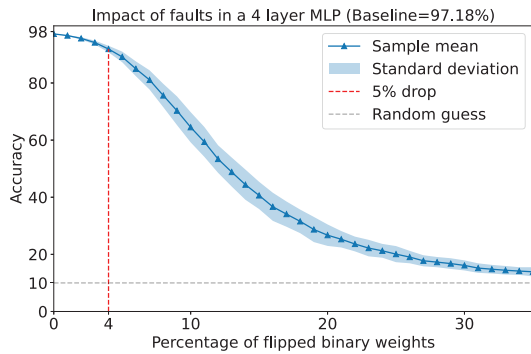


Fig. 1: Monte carlo simulation (100 samples) of weight bit flips in a 4-layer neural network.

- Operates in a gray-box setting: no changes to the accelerator and only requires write access to model weights
- Evaluation on PYNQ-Z2 and ZCU104 to demonstrate consistent efficiency and portability to cloud FPGAs

This paper is structured as follows: first, the background is introduced in section II, followed by the voltage-based detection method described in section III. The experimental setup is detailed in section IV, with results presented in section V. Finally, the conclusions are summarized in section VI.

II. BACKGROUND

A. Fault Detection in NN Hardware

Effective detection of runtime faults in NN accelerators, is essential for maintaining reliability and functionality. The effect of such faults on model predictions is illustrated in Figure 1, which shows the degradation of prediction accuracy under random weight bit flips. Traditional methods for fault detection, such as Error-Correcting Codes (ECC) [23] and hardware redundancy techniques like Triple-Modular Redundancy [24] or Cyclic Redundancy Code (CRC) [13], have significant limitations. ECC methods effectively manage single-bit errors but do not cover faults in neural processing logic. Meanwhile, redundancy methods, though robust, result in substantial resource consumption, limiting their practicality for resource constraint scenarios.

Several methods have been proposed that specifically target fault detection in NNs. A secondary, redundant network is employed in [14] to detect discrepancies in the main model’s output. This method dynamically validates NN outputs during inference by comparing them with a lightweight replica trained under fault-free conditions. The main drawback is the overhead from maintaining and running the additional model. Activation Range Supervision is proposed in [15] to detect bit-flip faults in CNNs by observing neuron activation outputs. This approach requires modifying the model architecture, which limits compatibility with existing deployments.

A real-time detection and recovery approach against bit-flip attacks is presented in [25]. It leverages checksum-based 2-bit signatures calculated over interleaved and masked weight groups for fault detection. Similarly, Javaheripi et al. [16] proposed a detection mechanism for adversarial bit-flip attacks on DNN weights, by comparing hash signatures from sensitive

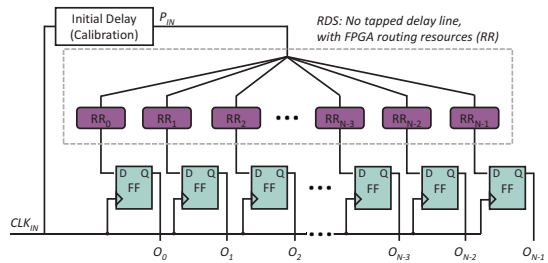


Fig. 2: Routing delay sensors as in [17]

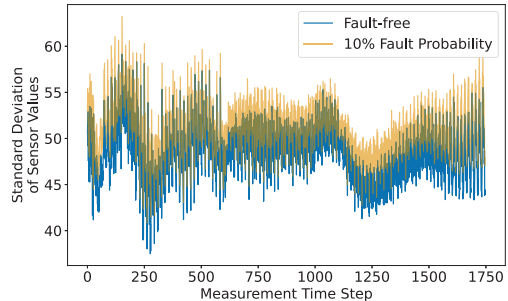


Fig. 3: Standard deviation of traces under 10% fault probability and fault-free operation.

layers with runtime hashes to identify integrity breaches. Both methods focus solely on detecting adversarial weight faults and require additional logic for signature verification.

In contrast to these methods, we target a gray-box setting without model duplication or exact knowledge about the hardware implementation. It enables real-time detection of both faults in model parameter memory as well as faults in neural processing logic and is tailored to binary neural networks, while introducing minimal hardware overhead.

B. Voltage Fluctuation Sensors

On-chip voltage fluctuation sensors, including delay-line-based Time-to-Digital Converters (TDCs) [26] and RDS [17], leverage the correlation between supply voltage and signal propagation delay to monitor real-time voltage stability. In FPGA implementations, TDCs traditionally use carry-chain or logic-element delay lines to convert voltage-induced delay changes into measurable timing shifts, achieving nanosecond-level resolution and often including multi-stage quantization with coarse-fine calibration. Ring oscillator (RO) sensors [27] provide simpler, calibration-free alternatives but offer lower sampling rates. We use RDS (see Figure 2) due to their demonstrated accuracy and robustness to temperature variation, outperforming other delay-based sensing approaches [17].

III. VOLTAGE FLUCTUATION BASED CED

A. Hypothesis and Overview

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 3 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 4.

B. Classifier Design and Training

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

1) *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.

2) *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

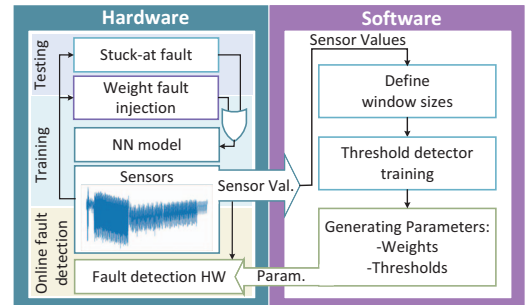


Fig. 4: Overview combining hardware and software: Faults are injected into the accelerator, and voltage traces are recorded via on-chip sensors. In software, these traces are used to train the detector. The resulting parameters are written to hardware.

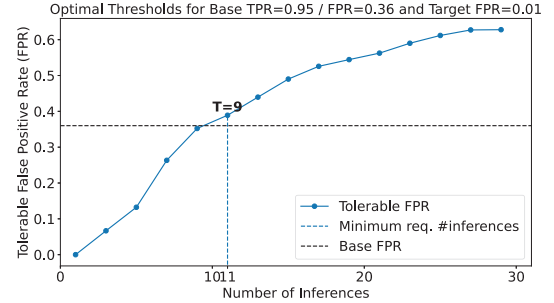


Fig. 5: Tolerable FPR as a function of the number of inferences. For a base TPR of 0.95 and FPR of 0.36, a threshold of $T = 9$ over 11 inferences reduces the effective FPR below the target of 0.01.

a global detection threshold to trigger a fault flag. During training, 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.

3) *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 sequence of inferences. Furthermore, detection over time is particularly advantageous for continuous inference

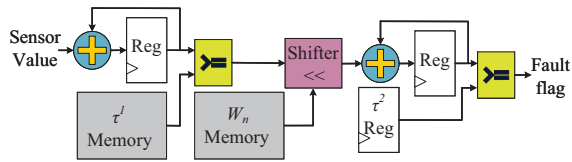


Fig. 6: Hardware block for per-window threshold fault detection. Window size and thresholds are fully configurable. Scores are aggregated and evaluated with minimal latency.

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 5 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.

C. Hardware Design

Our detection method is implemented as a standalone hardware block adjacent to the NN accelerator, as shown in Figure 6. 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 separation between static architecture and dynamic configuration supports a broad range of deployment scenarios with minimal overhead.

IV. 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.

A. Models and Datasets

Table I summarizes the NN models evaluated in this work. All models are binary, with 1-bit weights and activations, and trained using the Brevitas library [28]. The experiments include a fully connected Multilayer Perceptron (MLP) and a Convolutional NN (CNN), both trained on the MNIST dataset [29], as well as a VGG-like [30] model trained on a subset of the German Traffic Sign Recognition Benchmark (GTSRB) [31]. The MNIST MLP consists of three hidden layers, while the MNIST CNN uses three convolutional layers

TABLE I: 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

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.

B. Accelerator Implementation

We use the FINN framework [32], 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.

C. 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.

D. 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.

V. RESULTS

A. 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 7, the *fault coverage*, i.e., the percentage

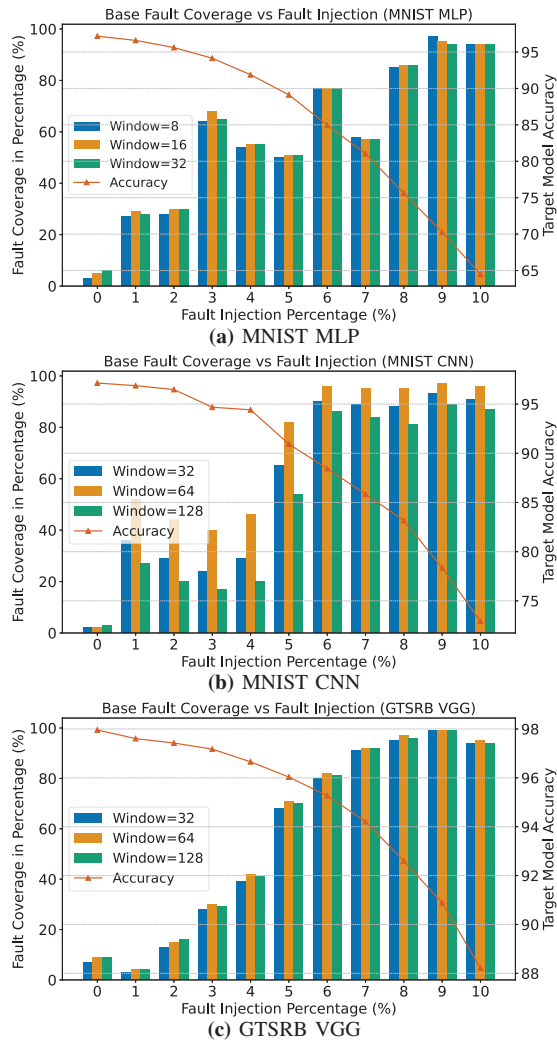


Fig. 7: Base fault coverage of the detector without detection over time. The target model accuracy with varying fault intensity shows the coverage depending on the inflicted accuracy degradation. From this analysis the optimal window size can be selected.

of correctly identified faulty instances, increases with fault intensity. The accompanying 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 8 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 ($< 0.5\%$).

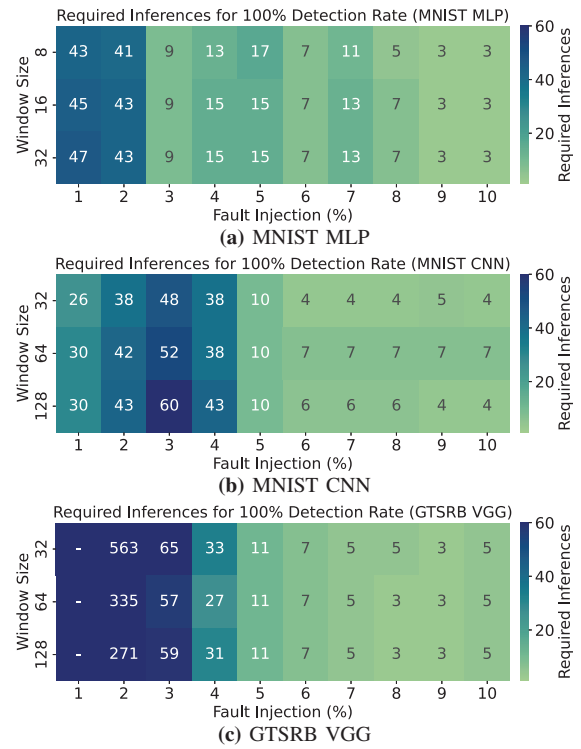


Fig. 8: Required #inferences for $>99\%$ weight bit-flip detection.

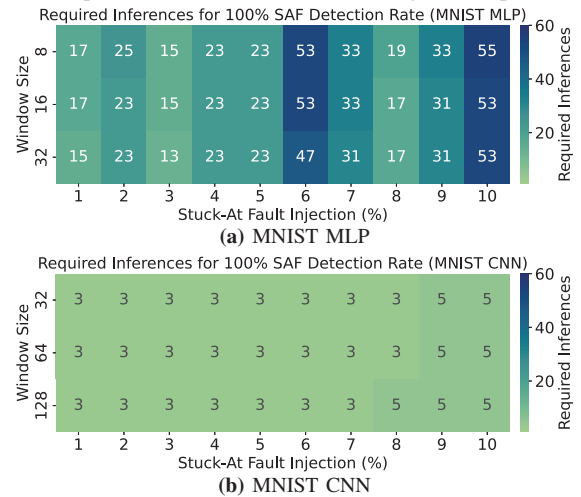


Fig. 9: Required #inferences for $>99\%$ logic fault detection. GTSRB is omitted, as it achieves 100% accuracy for any fault intensity.

B. Stuck-At Faults

Figure 9 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 9a), 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 9b) 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).

TABLE II: 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}

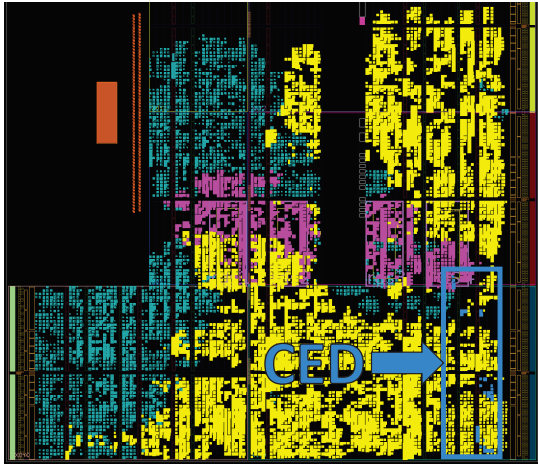


Fig. 10: Floorplan showing CNN (yellow), RDS sensors (pink), and our CED (blue, bottom right, highlighted by box).

TABLE III: FPGA Resource Utilization on PYNQ-Z2

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

C. Hardware Implementation

The full detection pipeline is implemented on the FPGA. Figure 10 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.

Table II 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 III, our detection logic consumes a negligible fraction of the FPGA 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.

D. Discussion

Our evaluation demonstrates that the proposed detector can be deployed in continuous inference settings without compromising responsiveness or throughput. As shown in Table II, 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.

VI. CONCLUSION

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.

ACKNOWLEDGMENT

This work is funded by the German Federal Ministry of Research, Technology and Space (BMFTR) in the framework of design tools for sovereign chip development with open-source (DE:Sign DI-EDAI).

REFERENCES

- [1] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.
- [2] Amazon, "Ec2 f1 instances."
- [3] Microsoft, "Deploy ML models to field-programmable gate arrays (FPGAs) with Azure Machine Learning," Apr. 2022.
- [4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 6, pp. 4909–4926, 2021.
- [5] S. A. Singh and K. A. Desai, "Automated surface defect detection framework using machine vision and convolutional neural networks," *Journal of Intelligent Manufacturing*, vol. 34, no. 4, pp. 1995–2011, 2023.
- [6] 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.
- [7] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.
- [8] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE access*, vol. 5, pp. 17322–17341, 2017.
- [9] 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.
- [10] 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.
- [11] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.
- [12] 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)*, pp. 1–5, IEEE, 2020.
- [13] P. Koopman and T. Chakravarty, "Cyclic redundancy code (crc) polynomial selection for embedded networks," in *International Conference on Dependable Systems and Networks, 2004*, pp. 145–154, IEEE, 2004.
- [14] 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*, pp. 101–112, 2020.
- [15] F. Geissler, S. Qutub, S. Roychowdhury, A. Asgari, Y. Peng, A. Dhamaia, 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.
- [16] 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)*, pp. 1–9, IEEE, 2021.
- [17] 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*, pp. 543–567, 2023.
- [18] D. Zhang, Q. Ren, and D. Su, "An on-chip path delay measurement sensor for aging monitoring," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, pp. 1–4, IEEE, 2021.
- [19] S. M. Ghasemi, J. Krautter, T. Gheshlaghi, S. Meschkov, D. R. Gnad, and M. B. Tahoori, "Degradation monitoring through software-controlled on-chip sensors for risc-v," in *2024 IEEE European Test Symposium (ETS)*, pp. 1–6, IEEE, 2024.
- [20] P. Chen, C.-C. Chen, C.-C. Tsai, and W.-F. Lu, "A time-to-digital-converter-based cmos smart temperature sensor," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, pp. 1642–1648, 2005.
- [21] J. Gravelier, J.-M. Dutertre, Y. Teglia, P. L. Moundi, and F. Olivier, "Remote side-channel attacks on heterogeneous soc," in *International Conference on Smart Card Research and Advanced Applications*, pp. 109–125, Springer, 2019.
- [22] 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)*, pp. 259–264, IEEE, 2023.
- [23] 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.
- [24] 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.
- [25] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, "Radar: Runtime adversarial weight attack detection and accuracy recovery," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 790–795, IEEE, 2021.
- [26] S. Henzler and S. Henzler, *Time-to-digital converter basics*. Springer, 2010.
- [27] K. M. Zick and J. P. Hayes, "Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 1, pp. 1–26, 2012.
- [28] A. Pappalardo, "Xilinx/brevitas," 2023.
- [29] 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.
- [30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [31] 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*, no. 1288, 2013.
- [32] 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," 12 2016.