IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2026, No. 1, pp. 448–471. DOI:10.46586/tches.v2026.i1.448-471

PWNN: Power-Wasting Neural Network As Remote Fault Injector

Huashuangyang Xu, Sergej Meschkov, Vincent Meyers and Mehdi Tahoori

Karlsruhe Institute of Technology, Karlsruhe, Germany
{huashuangyang.xu,sergej.meschkov,vincent.meyers,mehdi.tahoori}@kit.edu

Abstract. The explosive growth of AI-driven services has led to cloud-based Field Programmable Gate Array (FPGA) accelerators as key enablers of high-performance training and inference in modern data centers. Since 2024, the demand for deploying large AI workloads, especially Large Language Model (LLM), in the cloud has increased dramatically, intensifying competition among cloud providers and increasing pressure on shared FPGA infrastructures. This increasing reliance highlights the need for robust hardware security measures for cloud FPGAs. A particularly serious threat is fault injection attacks, which exploit dynamic voltage fluctuations to induce timing faults, potentially compromising functional integrity and bypassing cryptographic protections. However, existing verification procedures and structural Design Rule Check (DRC) remain blind to attacks embedded in benign-looking circuits. In this paper, we present Power-Wasting Neural Network (PWNN), a novel adversarial technique that leverages the inherent switching behavior of neural network operations to act as a power-waster circuit under adversarial input patterns. We systematically explore network architectures, and input patterns to craft configurations that induce voltage fluctuations capable of triggering timing faults for successful Differential Fault Analysis (DFA). Our PWNN implementation uses a standard open-source tool chain and passes all pre-implementation verification checks, while covertly inducing faults at runtime. We demonstrate on both the AMD ZCU104 and PYNQ-Z2 that PWNN can reliably cause timing faults on the critical path of a co-located AES-128 block cipher, enabling the rapid collection of correct/faulty ciphertext pairs needed for DFA-based key recovery. These results show that functionally correct, DRC compliant accelerators can serve as powerful, adaptive fault injectors that invalidate assumptions about bitstream security and hardware isolation.

Keywords: Fault injection attack · Neural network · AES key recovery · Cloud FPGA

1 Introduction

Modern computing infrastructures, including AI applications [FAFK+24, OLK25], online banking [YWC+24], healthcare systems [HBH24, TCC+24], industrial control and communication networks, form the foundation of almost every aspect of daily life. Ensuring the security of both software and hardware platforms is critical. While software vulnerabilities have been extensively studied and patched for decades, hardware is often assumed to be inherently trustworthy once manufactured and configured. However, as hardware functions become increasingly complex and reconfigurable, new attack vectors are emerging that can undermine this assumption [CNG+24, SZY21, LHWW22].

Field Programmable Gate Array (FPGA) is playing an increasingly important role in modern computing. Its ability to be reconfigured on demand makes FPGAs ideal for accelerating a variety of workloads, including AI inference, cryptographic processing

Received: 2025-07-15 Accepted: 2025-09-15 Published: 2026-01-16

and real-time signal analysis. In recent years, cloud service providers have integrated FPGAs into large data centers to enable high-performance, low-latency computing for a wide range of users. Platforms such as Amazon EC2 F2 [Ama25] and Microsoft Azure's [Mic25] configurable compute instances allow customers to deploy custom logic in a shared hardware environment. This multi-tenant FPGA model offers flexibility and efficiency, but also brings unique security challenges that arise from the physical sharing of on-chip resources [EKC19]. In addition, advances in heterogeneous and 3D integration have enabled FPGAs to be tightly integrated with CPUs, GPUs, and DRAM in the same package, forming chiplet-based SoCs [Man25, CZL+25, ZYCH22, ZWY+24]. For example, AMD's recent designs leverage chiplet architectures to incorporate reconfigurable logic (Xilinx XDNA) alongside general-purpose cores and high-bandwidth memory, making FPGAs an integral component of unified compute fabrics [AMD24, AMD25]. These trends amplify the attack surface, as FPGAs become embedded in high-performance systems with shared power, thermal, and memory domains, raising the stakes for hardware-level security in both edge and cloud deployments.

In cloud FPGA environments, users occupy separate partitions that are logically isolated but share underlying resources such as the PDN, interconnect, and clocks. This shared infrastructure enables malicious tenants to compromise the timing or correctness of neighboring designs One notable vulnerability is voltage fluctuations caused by aggressive dynamic power consumption [MLPK20, PHT20]. A design that intentionally switches a large number of gates at a high frequency can cause significant transient local voltage drops that lead to timing faults in neighboring circuits.

Simple circuits such as ring oscillators or high-speed shift registers can act as power-wasters by deliberately drawing current to disrupt the critical timing path of other parts of the chip [KSY23, ZS25]. To defend against these attacks, FPGA vendors and cloud providers implement a combination of Design Rule Check (DRC), bitstream scanning, and enforcement of placement and routing constraints to detect and block known fault injection attack structures during the deployment of user designs [LMG+20, LPPK23, ZGK22, MSRZ+25]. However, these safeguards are primarily focused on detecting fixed, structural patterns and are often ineffective against adaptive or input-triggered fault injectors.

In this paper, we propose Power-Wasting Neural Network (PWNN) that re-purpose binarized neural network accelerators as dynamic power wasters that trigger precisely controlled fault injection attacks on cloud FPGA platforms. These designs leverage the inherently high toggle rates of binary operations—such as XNOR based matrix multiplication and popcount accumulation that are commonly found in lightweight AI inference engines. Under typical operation, the network appears benign, functioning as a standard low-precision neural model. With specially developed trigger inputs, however, the same operations generate extreme switching activity that leads to abnormal voltage drops on Power Distribution Network (PDN). This dual behavior makes the network appear structurally innocent, as the malicious behavior is only activated conditionally. Because the PWNN is built exclusively from standard lookup table logic and adheres to all synthesis constraints, it is assumed to pass manufacturer verification and escapes existing security checks in the cloud.

We validate our approach mainly on two AMD FPGA platforms: the resource-constrained PYNQ-Z2 and the commonly used cloud FPGA Zynq UltraScale+. Our experiments show that the activation of PWNN can reliably inject single-bit or multi-bit faults in a co-located AES-128 block cipher [Nat23]. This demonstrates that it is possible to collect correct and faulty ciphertext pairs for Differential Fault Analysis (DFA) [DLV03] in under a second with a few plaintexts under certain conditions. We also systematically investigate how factors such as the network topology and the number of concurrent PWNN instances influence fault rate and injection accuracy. In this work, we model PWNN architectures using an open-source FINN framework from AMD research lab [UJFG+17].

Our Contributions

Our main contribution is the introduction of PWNN, a power-wasting neural network designed for remote fault injection on cloud FPGA platforms. We propose a methodology that leverages begnig-looking AI workloads to induce faults while evading detection. PWNNs convert binarized neural network accelerators into stealthy circuits that behave normally under random inputs but induce excessive power consumption when activated by specific trigger inputs, all while remaining fully compliant with vendor design rule checks and bitstream security verification. Unlike other power-wasters presented in previous research [MLPK20, PHT20] PWNNs are not statically identifiable by security check in cloud FPGA platform. They can only be triggered at runtime by certain input patterns, which makes them extremely difficult to detect by existing security measures.

Furthermore, we also conduct a comprehensive and systematic study of PWNN design parameters. We investigate how the depth of the adder tree, the number of parallel PWNN instances and their placement on the FPGA fabric influence the switching activity, the fault intensity and the rate of exploitable faults. This systematic investigation shows how design decisions influence the effectiveness of the attack.

Finally, we validate our approach through practical experiments targeting both cryptographic and arithmetic modules. In particular, we show that PWNNs can reliably inject faults into the AES-128 encryption module and the 256-bit Ripple Carry Adder (RCA) on two representative AMD FPGA platforms: the resource-constrained PYNQ-Z2 and the commonly used cloud FPGA Zynq UltraScale+. Our results show that PWNNs can reliably generate correct and faulty ciphertext pairs under precise control, enabling a range of fault attacks on cryptographic cores. While we demonstrate DFA as one concrete example, the underlying fault injection attack capability of PWNNs is broadly applicable to many fault-based attack strategies.

2 Background and Related work

2.1 Threat Model

We examine a well-documented multi-tenant FPGA environment [KGT18, DSZ21], which includes applications within data centers such as the Amazon EC2 F2 FPGA instance [Ama25], as well as SoC platforms. In these scenarios, distinct CPU processes can independently utilize portions of the FPGA logic, as illustrated in Figure 1. The victim and adversary operate within separate processes managed by the same operating system and each have access to a dedicated fraction of an FPGA, where they can load their own arbitrary designs, such as cryptographic accelerators. Their respective designs are logically and spatially isolated on the FPGA, but it is assumed that the entire FPGA fabric shares a common power supply.

Further, we assume that the victim employs their allocated region for implementing a security-critical algorithm, such as a block cipher, with a secret key either hard-coded or loaded at runtime. In this work, we focus on AES as the target algorithm. The adversary is assumed to have the ability to issue arbitrary encryption requests and observe the corresponding outputs. In practice, it may suffice to trigger the encryption of the same (but random) plaintext twice, which is sufficient for DFA. Even weaker conditions may be adequate for other types of fault attack.

2.2 Power-Wasters Circuits and Fault Attacks on FPGAs

FPGAs in multi-tenant systems can be used by adversaries to exploit the vulnerability of shared PDN. In these attacks, a malicious circuit, which is usually called a power-waster, can be implemented on FPGA to force the shared supply voltage to drop or glitch, resulting

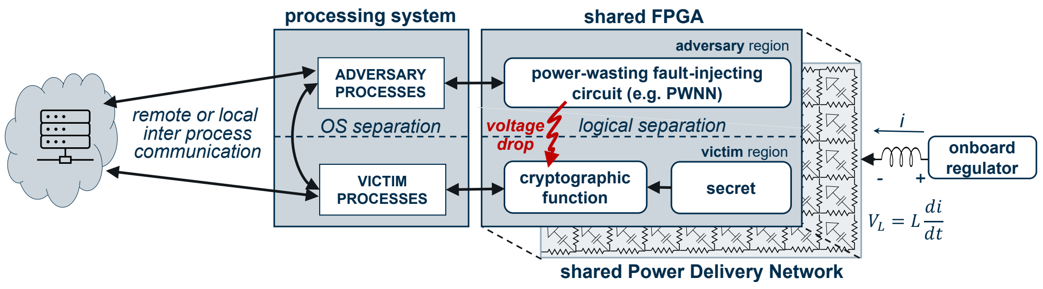


Figure 1: Threat model in multi-tenant environment.

in timing faults in victim logic [KGT18, MDH+22]. The adversary can then use the faulty outputs (for example, from an AES cipher) to perform DFA or other key-recovery methods. Such attacks are possible on-chip within single FPGA [KGT18] or FPGA as part of a System on Chip (SoC) [MDH+22], or theoretically even with multiple chips sharing power supply at the board level.

2.2.1 Effect of Power-Waster Circuits on FPGAs

The dynamic power consumption of a (power-waster) circuit in an FPGA is primarily determined by switching activity following the relation, where $C_{switching}$ is the capacitive load , $f_{switching}$ is the operating frequency, and V_{DD} is the supply voltage:

$$P_{dyn} \approx V_{DD}^2 \cdot f_{switching} \cdot C_{switching} \tag{1}$$

Circuits designed to maximize signal toggling with preferably minimal logic utilization are effective for inducing high dynamic power consumption in FPGAs. The main idea here is to create events with peak dynamic power consumption which will not be well regulated at PDN and cause a voltage drop. For instance, it was shown that simultaneous activation of a large amount of Ring Oscillators (ROs) causes the voltage to drop by a certain amount and then more slowly return to the original value [GOKT18]. Such voltage drops can be high enough to crash the whole FPGA-based system [GOT17].

To explore how the supply voltage of an FPGA can be reduced using on-chip logic elements, it is essential to understand the behavior of the PDN under the influence of different circuit designs. The PDN spans from the external voltage regulation module on the board to the internal power rails and ultimately to every transistor within the FPGA. It is typically modeled as a mesh comprising resistive, inductive, and capacitive components. Consequently, the supply voltage is affected by two primary factors: the average static current drawn by the implemented design (resulting in an $I_{static} \cdot R$ drop), and transient voltage fluctuations caused by switching activity and inductive effects. This relationship is described by the Law of Inductance:

$$V_{drop} = I_{static} \cdot R + L\frac{di}{dt} \tag{2}$$

However, with ongoing technology scaling, the impact of static voltage drops has diminished in significance, while the influence of dynamic voltage fluctuations due to inductive components and switching activity has become increasingly prominent.

Further, the runtime variations of voltage cause changes in actual signal delays in the implemented circuits on the FPGA, which can be described with the following relation:

$$t_{vdrop\ delay} \approx \frac{1}{V_{DD} - V_{drop}} \tag{3}$$

The clock signal driving any functional block in a synchronous FPGA design imposes timing constraints on all associated signal paths. Typically, FPGA mapping tools ensure these constraints are met by accounting for expected process variations as well as runtime fluctuations in voltage and temperature. However, under extreme voltage glitch conditions, these assumptions may no longer hold, potentially leading to timing violations. This can be described with the following relations, where t_{clk} is the clock period, t_{setup} is the register setup time, and $t_{comb\ static}$ is the combinational path delay excluding runtime variations:

$$t_{clk} < t_{setup} + t_{comb\ static} + t_{vdrop\ delay} \tag{4}$$

As a result, incorrect data may be latched into sequential registers, causing functional faults or exploitable faults.

2.2.2 Known Power-Waster Circuits for FPGAs

The most common and simple power-waster circuit is an array of ROs. In an FPGA the RO is implemented as LUT-based combinational loops, which toggle the output signal, drive it back to the input and oscillate this way with a very high frequency. For fault injection attack, an array of such ROs is toggled at high frequency. The excessive switching current causes a supply voltage drop, lengthening logic delays and triggering timing faults. To make the attack more precise, the number of active ROs, activation patterns, frequency and duty cycles can be controlled by the adversary [KGT18, PHT20].

Additionally, power-wasters can be implemented with shift registers driven by a very high clock speed. Here, a set of looped shift registers is used, and each is initialized with a sequence of 1's followed by 0's in a way that after each shift the signal at the output of each register in the chain is toggled [PHT20]. Although shift registers are less effective at wasting power compared to RO-based circuits, they are often integrated with the functional logic of an Intellectual Property (IP) core, making them difficult to distinguish from legitimate design elements. This allows a malicious user to embed numerous shift registers within an IP core to stealthily increase power consumption and potentially induce voltage instability.

Since signal glitches contribute significantly to a circuit's overall dynamic power consumption, they are also exploited in the design of power-wasting circuits. In this context, "glitches" do not refer to fault injection attacks, but to unwanted multiple output transitions of a logic gate caused by mismatched arrival times of its input signals. These mismatched arrival times typically arise from imbalanced path delays leading to the gate's inputs. Furthermore, the effect of glitches can be magnified in longer combinational paths, which are also designed in a way to make many concurrent and consecutive unbalanced paths. An example is the combinational concatenation of a single block-cipher round instances (such as Advanced Encryption Standard (AES)) repeated in a longer chain with XOR gates in between the instances driven by the outputs of different stages in the chain [PHT20]. Another example is s1238 from the ISCAS'89, which also can be used as a power-wasting circuit as shown in [KGT21].

2.2.3 Detection of Power Wasters

Design Rule Check (DRC). Cloud providers rely on FPGA vendor DRC to catch naive Trojans. For instance, Amazon's F1 service runs AMD DRC on the user's netlist to reject obvious ROs before generating the bitstream [ZVF+21]. Gross short-circuits or extremely high-fanout nets are also flagged. However, as noted in the literature, DRCs are limited: a ring oscillator with a register or latch added (making it sequential) may bypass the "no feedback loop" rule yet still toggle at high frequency [PHT20]. Similarly, glitch-amplifier circuits (e.g. a flip-flop feeding an XOR with delayed feedback) can self-oscillate without forming a simple loop.

Signature/Pattern Scanning (Bitstream Antivirus). Bistream can be scanned for malicious signatures or patterns like it is done by conventional antivirus software [KGT19, LMG+20]. This works well for any known malicious circuit designs, including power-wasting circuits. However, it would require continuous updates to protect against any newly introduced malicious designs.

Machine Learning on Bitstreams or Netlists. Machine Learning (ML) can be trained to recognize malicious patterns in Bitstreams or Netlists. Convolutional Neural Network (CNN) trained on raw bitstream data can distinguish normal vs. malicious designs [CC22]. Graph Neural Network (GNN), the MaliGNNoma framework [ANK+24], builds a graph of the synthesized netlist (gates and interconnects) and applies a Graph Neural Network to classify it as benign or malicious.

Formal and Analytical Methods. Run formal checks (SAT/SMT) for forbidden structures on the netlist. For instance, one could use a SAT solver to detect any combinational loop or use graph algorithms to find exceptionally high-fanout nodes. These methods are akin to vendor DRCs but can be more exhaustive. However, they share the same limitation: they must know what to look for.

Runtime Monitoring. FPGAs include sensors that can indirectly detect anomalies at run-time. For example, AMD Ultrascale+ parts have on-chip power monitors and RO-based sensors; Intel Stratix devices expose power rails and temperature sensors. Additionally, sensors can be implemented using reconfigurable FPGA logic, e.g. based on Time-to-Digital Converter (TDC) [MZTF22]. These can be polled to flag unusual switching activity.

In modern FPGA design flows, vendor toolchains (e.g., Vivado, Quartus) provide not only synthesis/P&R/timing sign-off but also built-in power analysis. These analyzers come in two modes: vectorless (default) and vector-based (simulation driven). Vector-based estimates are more accurate but cannot guarantee coverage, while advanced statistical methods (e.g., sigmaDvD) are not yet integrated in current FPGA flows. Estimating worst-case power is especially infeasible for NN accelerators in the cloud, since inputs and weights can be modified at runtime, enabling adversaries to maximize switching. Vendor DRCs enforce legality and block obvious oscillators, but they cannot capture NN parameter dependent power peaks. This gap makes multi-tenant cloud FPGAs vulnerable: attackers can deliberately stress the shared PDN to impact co-located designs.

2.3 FINN

FINN [UJFG+17] is an open-source framework developed by AMD Xilinx Research for generating FPGA accelerators from quantized and binarized neural networks (QNNs and BNNs).Its architecture adopts a dataflow model in which each neural network layer is implemented as a separate, stream-connected hardware block, enabling fine-grained control over latency, throughput, and parallelism. FINN leverages High-Level Synthesis (HLS) to generate vendor-compatible IP cores and supports exporting trained models from PyTorch (via Open Neural Network Exchange (ONNX)) into deeply pipelined, hardware-efficient architectures.

The accelerators generated by FINN are structured to be compatible with standard vendor tool chains like Vivado, which perform DRC during implementation. Our PWNN designs, generated through FINN's flow, pass all vendor-enforced checks without triggering alerts, demonstrating that such adversarial behavior can be embedded even in tool chain compliant designs.

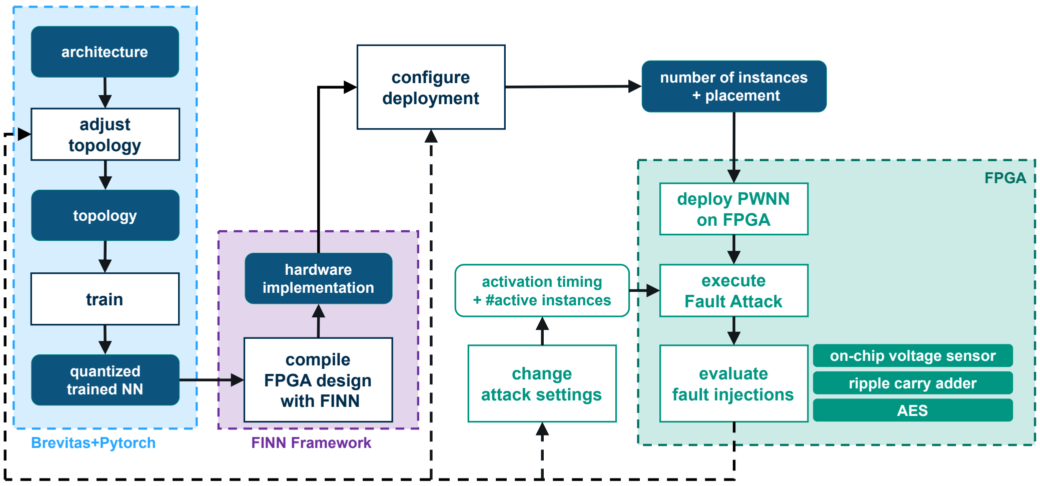


Figure 2: PWNNs are iteratively optimized by evaluating values from on-chip voltage sensor, fault intensity in Ripple Carry Adder (RCA) and fault rates from Advanced Encryption Standard (AES), guiding both network architecture exploration and input/weight optimization.

3 Methodology

3.1 End-to-End Workflow

In this subsection, we present our method for constructing a malicious Binarized Neural Network (BNN) that functions as a power-wasting circuit. The goal is to induce voltage drops in neighboring circuits on multi-tenant FPGA platforms, causing timing faults when triggered. We leverage AMD's open-source FINN framework [UJFG+17] to produce accelerators that appear benign but exhibit harmful, input-dependent behavior at runtime. As part of the standard design flow, we ran the DRC using the Vivado Design Suite to ensure our accelerators complied with implementation constraints. Our attack consists of four key stages:

1. **Topology Search & Optimization**: We systematically explore a range of BNN topologies by varying the number of input features, the size of hidden layers, and the activation quantization strategies, with the goal of maximizing switching activity while maintaining a small hardware footprint. Based on an empirical evaluation, we choose a compact single-layer MLP with binary weights and activations (± 1) implemented with Brevitas and Pytorch [Xil19] in software.

2. **Joint Input & Weight Optimization**: We jointly optimize the network weights and two input images using a custom loss that maximizes the Hamming distance between XNOR outputs and bit flips in the adder tree. Alternating these inputs each clock cycle drives near-maximum toggling across all neurons, thereby maximizing switching activity.

3. **Stealth Compliant Synthesis via FINN**: The optimized model and input patterns are exported to ONNX and compiled with FINN into vendor HLS modules and streaming kernels. FINN produces a standalone bitstream that can be deployed with a victim core on a shared cloud FPGA. Although PWNN and victim logic are isolated in logic and placement, they still share low-level resources such as the PDN, clock trees, and interconnects.

4. **Cycle-Level Attack Execution**: At runtime on a cloud FPGA the adversary loads two input patterns into on-chip buffers and alternates them each inference to induce voltage drops on the shared PDN. These drops can produce timing faults on the victim's critical path at arbitrary cycles, enabling runtime fault attacks (e.g., DFA) to recover block cipher keys.

We propose a closed-loop design and deployment methodology for PWNNs, where fault effects are measured and fed back to guide the generation of power-wasting behavior that induces voltage-based timing faults. The workflow, shown in Figure 2, comprises two phases: *(i)* generation and optimization of the PWNN and *(ii)* fault-injection evaluation on a cryptographic target.

To identify an effective architecture, a broad set of BNN topologies are examined by varying layer width and input dimensionality. Each candidate is implemented in software using PyTorch and Brevitas, then subjected to joint input and weight optimization using the Adam solver [KB14]. During backward propagation we apply the loss function as defined in Subsection 3.2, updating both the network parameters and the two binary trigger images so that the internal circuit logic maximizes bit-level toggling during operation. Each candidate is compiled with the standard FINN framework tool chain, deployed to the FPGA alongside an n-bit RCA as a reference canary circuit, and exercised while an on-chip sensor records the resulting voltage drop. We select the inputs for the RCA in a way that forces the carry to propagate through each full adder in turn, the structure behaves like a long delay line: a deeper voltage drop stops the carry earlier, so the distance it travels, reflected in how many higher-order sum bits become faulty as the fault moves toward the most significant bit, is a direct measure of the magnitude of the voltage drop. In addition, we study the correlation between the bit-level fault pattern and the power trace to evaluate each design. If the induced fault intensity remains below the target value, the BNN topology needs to be adjusted, and the optimization loop is repeated. The optimization ends as soon as the fault intensity has reached the expectation within the resource and software limits. The resulting PWNN bitstream and the associated trigger inputs are then saved as the final configuration.

After completion of this phase, we evaluate the fault injection rates of the optimized PWNN for AES-128 block cipher on two distinct boards, namely an AMD PYNQ-Z2 and an AMD ZCU104. For each platform, we measure three metrics: the overall fault rate, the subset of faults that meet the classical DFA model, and the effort required to gather enough correct/faulty ciphertext pairs (number of plaintexts and consumed time). For experiments on the ZCU104, we deploy 8 instances of the PWNN and vary the number of active power-wasting accelerators running in parallel. Here we also elaborate on the effects of the placement and for a given number of activated PWNN instances, try all possible combinations of available PWNNs.

3.2 Generating Power-Wasting NN with FINN

We first train our PWNN in software using the Brevitas library developed by the AMD AI Research Lab. Brevitas is specifically designed to train quantized neural network models suitable for efficient hardware deployment while preserving the full functionality and flexibility of PyTorch. In contrast to typical use cases where throughput is often traded off for reduced area by folding the architecture, we configure the number of Processing Element (PE) and Single Instruction Multiple Data (SIMD) lanes to their maximum values, thereby ensuring full spatial parallelism in the hardware. This setup allows us to directly control the switching behavior at the granularity of individual XNOR and adder gates, resulting in predictable and reproducible power characteristics.

We choose the BNN as the base architecture for our PWNN method for two main reasons, though other neural network architectures could also be extended or utilized

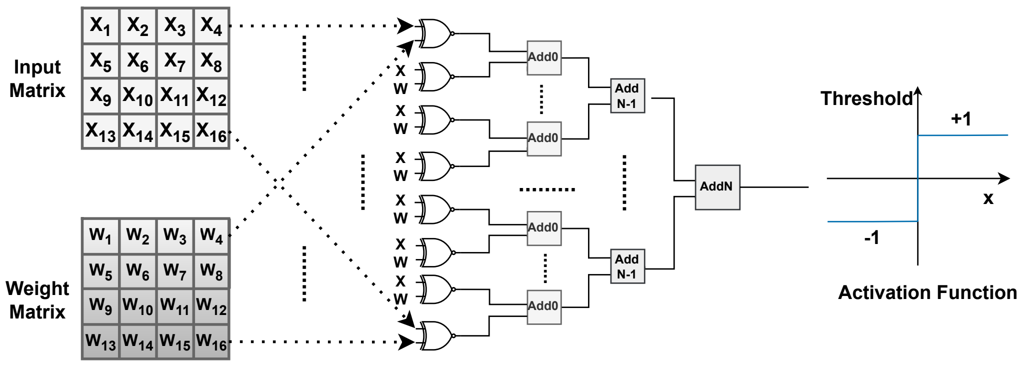


Figure 3: Resource-efficient BNN design in FINN: multiplications are implemented via XNOR-popcount logic, and nonlinear activations are realized as simple, threshold functions in hardware.

within the PWNN framework. First, the FINN framework efficiently saves hardware resources by implementing multiplications as XNOR followed by popcount operations, which aligns naturally with BNN computations. Second, due to the binary nature of BNNs, we can create a training target that explicitly controls both the input and weight patterns, giving us precise control over switching activity and thus power consumption variations within the FPGA fabric. However, once the network is deployed on our BNN accelerator, switching is controlled almost exclusively by the input patterns, as the XNOR logic renders the stored weights largely inactive.

Figure 3 illustrates the hardware data path of a single neuron: individual bits of the input and weight vectors are compared via XNOR gates, and their outputs are fed through a deep adder tree that accumulates the total popcount. Accumulated result is then evaluated by a simple threshold function, which produces a binary activation of either -1 or $+1$.

To generate PWNN configurations that maximize dynamic power consumption on FPGA hardware, we develop a custom training procedure that jointly optimizes both the weights and the selected inputs of the network. The goal is to generate input pairs that trigger high density gate-level switching activity in the hardware, thereby inducing frequent and widespread bit-flips during inference. As detailed in algorithm 1, we optimize the weights θ and two binary inputs x_1, x_2 simultaneously. For each training step, the network is executed on both inputs, resulting in the corresponding bitwise XNOR outputs and popcount intermediate values on each layer, which together capture the internal switching activity of PWNN.

The loss function maximizes the absolute differences between these outputs:

$$\mathcal{L} = -\|a_1 - a_2\|_1 - \|p_1 - p_2\|_1,$$

Here, $\mathcal{L}$ measures the total number of bits that toggle between two successive forward passes both at the XNOR gate outputs and at intermediate results of the adder tree. Rather than minimizing a conventional loss, the training objective is to maximize this switching activity. In effect, the optimizer searches for network parameters that induce the greatest number of bit flips across the entire circuit, thereby driving the model toward configurations that exhibit maximal internal dynamic behavior.

Optimization proceeds via the Adam optimizer, with Straight-Through Estimators (STE) used to back propagate gradients through binarized operations. After each update, weights are clipped to $[-1, 1]$ and binarized, and inputs are clipped to $[0, 1]$ to maintain valid binary states. The best-performing configuration (θ^*, x_1^*, x_2^*) is retained.

Algorithm 1: Co-optimization of inputs and binarized weights for PWNN inference, driving peak switching activity in hardware.

```
   Input: Quantized BNN model $\mathcal{F}_\theta$;
   Random binary inputs $x_1, x_2 \in \{0,1\}^{h \times w}$;
   Learning rate schedule $\{\eta_t\}$; total epochs $N$
   Output: Optimized weights $\theta^*$ and inputs $x_1^*, x_2^*$
1  Notation:
2  - $a_i$: bitwise XNOR output for input $x_i$
3  - $p_i$: intermediate bit-flip counts in the adder tree during popcount stage for $a_i$
4  Initialize $\theta, x_1, x_2$ randomly ;
5  for $t = 1$ to $N$ do
6      $(a_1, p_1) \leftarrow \mathcal{F}_\theta(x_1)$ ;
7      $(a_2, p_2) \leftarrow \mathcal{F}_\theta(x_2)$ ;
8      $\mathcal{L} \leftarrow -\|\text{sign}(a_1 - a_2)\|_1 - \|\text{sign}(p_1 - p_2)\|_1$ ;
9      Compute gradients $\nabla_{\theta, x_1, x_2}\mathcal{L}$ ;
10     Update:
           $\theta \leftarrow \theta - \eta_t \nabla_\theta, \quad x_1 \leftarrow \text{clip}(x_1 - \eta_t \nabla_{x_1}, 0, 1), \quad x_2 \leftarrow \text{clip}(x_2 - \eta_t \nabla_{x_2}, 0, 1)$
       Clip weights: $\theta \leftarrow \text{clip}(\theta, -1, 1)$ ;
11     if loss improves then
12         Save $\theta^*, x_1^*, x_2^*$
13 return $\theta^*, x_1^*, x_2^*$
```

Finally, this configuration is deployed via the FINN tool chain into a fully parallel accelerator (PE=MAX, SIMD=MAX), ensuring that the toggling behavior optimized in training manifests faithfully in hardware, resulting in significantly increased dynamic power consumption and effective fault injection attack.

In our FPGA implementation generated by the FINN framework, the accelerator IP is originally designed with a direct streaming interface where input pattern flows from the input Direct Memory Access (DMA) directly into the stitched accelerator block. To enable more controlled and flexible input pattern generation for fault injection attack experiments, we modify this data path as follows:

- We break the direct connection between the input DMA and the accelerator IP block.
- In place of the direct streaming path, we insert a custom controller module with on-chip memory.
- The ARM processor writes input patterns into the controller's on-chip memory.
- The controller then reads out these pre-stored input patterns into the accelerator, alternating between two distinct input patterns.
- Additional delay cycles used to activate PWNN can be configured via AXI registers in controller.

By switching between two input patterns stored on the chip, the PWNN is forced to repeatedly change its internal states, maximizing the voltage drops to add larger delays on the critical timing path and cause timing faults on the target module.

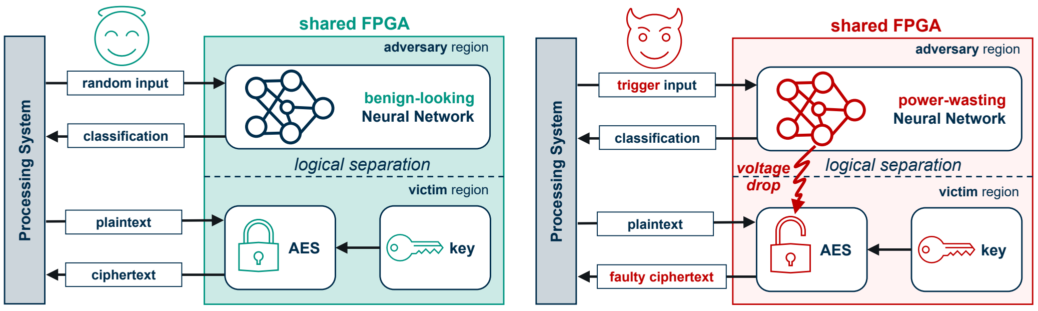


Figure 4: Example attack scenario evaluated in this work: On the left, a malicious NN, which acts like benign while processing random inputs. On the right, the same NN which turns into power-wasting circuit while processing a trigger input. This causes a voltage drop on the shared PDN, which result in delay faults at the logically separated neighboring AES-128 core, enabling key extraction via fault injection attack.

3.3 Attack Flow

In our example attack flow as illustrated in Figure 4, an adversary first uploads a malicious FPGA bitstream implementing the PWNN accelerator and deploys it into its own reconfigurable partition. To avoid triggering any dynamic power consumption alarms, PWNN initially runs on random input patterns and generates only nominal gate-level switching activity. As soon as the cryptographic module is in place and starts working, the adversary switches the PWNN input to two specially generated patterns that force maximum switching of the internal circuits and insert timing faults at certain clock cycles.

Next, the adversary uses the cloud provider's host API to transmit a set of selected plaintexts to the victim's AES engine and records the correct ciphertexts in stealth mode. With this data in hand, the adversary uses simple delay loops or inactive clock domain switching on the soft CPU to gradually match the highly active PWNN bursts with the critical, targeted round S-box computations of the AES kernel. By performing a small number of test injections and statistically analyzing the outputs, the exact cycle offset for fault induction is determined.

Once PWNN and the victim's AES-128 core are synchronized, the adversary first feeds a fixed set of selected plaintexts into the AES engine and records the corresponding correct ciphertexts under normal performance conditions. Then with help of the previously determined delay offset, the adversary reuses the same set of plaintexts while activating PWNN with maximum switching activity under high power input patterns. This leads to timing violations in the timing-sensitive part of the AES core and causes it to output the matching faulty ciphertexts for this identical plaintext sequence.

Ideally, we expect a one-byte fault to occur in round 8 of the AES computation in a precise fault injection attack with PWNN. This single-byte fault is then propagated to other rounds due to the MixColumns and ShiftRows operations, which distribute the single-byte fault across multiple state bytes. The adversary focuses on the last round and collects 4 different faulty ciphertexts, each generated by the same fault location but with a different plaintext.

With these four pairs of correct and faulty ciphertext, the adversary performs a DFA entirely in software: For each byte position, they first apply the inverse ShiftRows and inverse MixColumns from the last round of AES to isolate the affected byte. Then, for each of the 256 possible key byte values, they simulate the corresponding faulty SubBytes output and compare it to the observed error. Each of the four faulty patterns excludes the vast majority of incorrect candidates, and the intersection of the surviving guesses across all four patterns yields the exact key byte. This DFA procedure is repeated for all sixteen byte positions, adversary can fully reconstruct the 128-bit AES key.

During the attack, PWNN processes a specific pair of patterns which lead to high dynamic power consumption and pressure on the shared PDN, resulting in a voltage drop and timing violations in critical paths. While the cloud platform can detect this via board-level power monitoring systems, it has no way to isolate just the adversary's partition. The only available response is to reset the entire FPGA, disrupting all tenants. This makes mitigation costly and impractical, so the attack may be successful before any action is taken.

4 Results

4.1 Experimental Setup

To evaluate our proposed fault injection method described in Section 3, we conduct experiments on two AMD FPGA development platforms: PYNQ-Z2 and ZCU104. These platforms represent different resource scales, allowing us to assess our approach in both constrained and more capable environments. The PYNQ-Z2, based on a Zynq-7000 SoC, is used to study fault injection under limited hardware resources and serves as a baseline for minimal PWNN deployment. The ZCU104, featuring a Zynq UltraScale+ MPSoC, provides significantly more programmable logic, enabling us to explore the scalability of our method by varying the number and placement of PWNN instances. On this platform, we examine how duplicating multiple PWNN units and controlling their spatial layout impacts overall fault injection effectiveness.

We conduct two experiments to evaluate the effectiveness of our PWNN-based fault injection approach. In the first, we target a 256-bit RCA on an FPGA platform and use four on-chip voltage sensors to monitor fluctuations induced by the PWNN during execution, supporting architectural optimization. In the second experiment, we target an AES-128 block cipher deployed on both platforms, with the core operating at 210 MHz on the PYNQ-Z2 and 460 MHz on the ZCU104. We demonstrate practical fault injection within a complete system setup, which includes DMA-based data transfers and a custom controller for managing communication between the processing system and programmable logic.

4.2 Influence of Network Topology on RCA Fault Susceptibility

To understand how PWNN architectural parameters influence fault intensity before applying to real cryptographic targets, we first analyze a simpler arithmetic RCA module. The PWNN generated by the FINN compiler and evaluated in our experiments requires 10 clock cycles to complete each inference. To identify the clock cycles with peak switching activity in our single layer binarized PWNN and evaluate how neural network architecture influences fault intensity, we conduct fault injection attacks targeting the RCA. The critical path in the RCA is typically the carry bit chain, which on FPGAs is often implemented using two LUT5 resources per full adder, as illustrated in Figure 5. In a full adder, the critical path extends from the data inputs to the carry out output, traversing a sequence of XOR and AND/OR gates, and represents the longest delay within the adder. When multiple full adders are cascaded to form a RCA, the carry out from each stage feeds into the carry in of the next, creating an extended carry propagation chain. Thus, the critical path of an n-bit RCA starts at the least significant bit (LSB) inputs and propagates through the entire carry chain to the most significant bit (MSB) outputs. We exploit the timing sensitivity of the RCA by placing it adjacent to our power-wasting neural network, which induces localized voltage fluctuations through high switching activity. By flipping a single bit at the least significant position of the carry chain immediately before

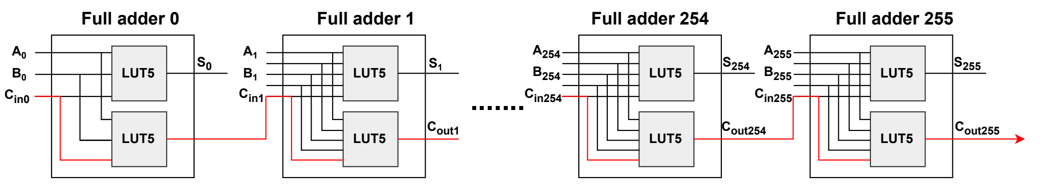


Figure 5: Critical path in full adder and carry bit propagation in 256-bit RCA.

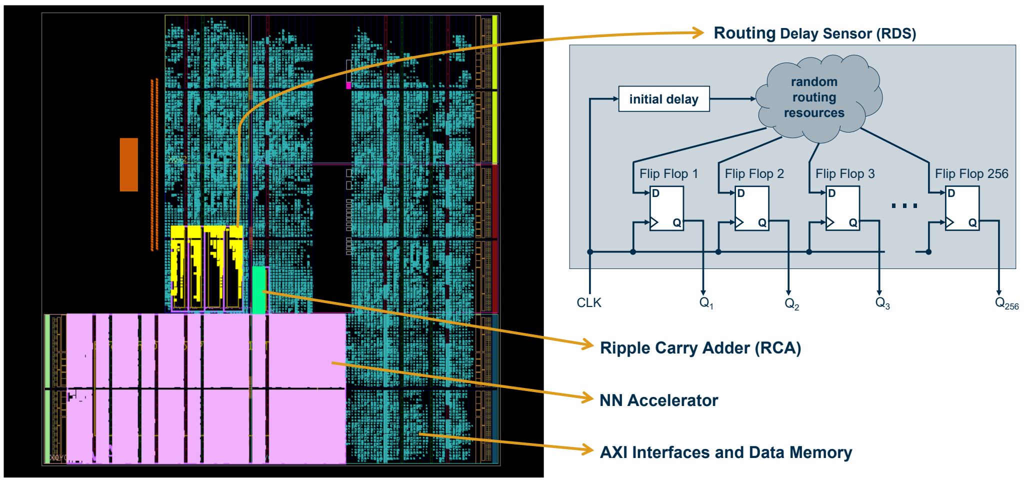


Figure 6: Hardware floorplan on the AMD PYNQ-Z2 board highlighting the positions of four voltage sensors, the RCA, and the Matrix-Vector Activation Unit (MVAU) used for a single fully connected layer in an Multilayer Perceptron (MLP).

propagation, we selectively stress the critical path of the adder. One input operand is held at all ones and the other carries a single '1' to maximize ripple effects. To better observe the fault manifestations induced by distinct PWNN architectures, synthesis constraints are applied to suppress timing violations and prevent synthesis tools from optimizing the structure of the target RCA.

4.2.1 Floorplan and On-Chip Voltage Sensing

Figure 6 shows the floorplan of our deployed system on the PYNQ-Z2 as generated by AMD Vivado software. We place four routing delay sensors near the RCA and above the MVAU of the accelerator. The Routing Delay Sensor (RDS) [SGS23] used in this experiment are more sensitive to voltage fluctuations compared to the TDC. Each sensor has a value range between [0–255] and is calibrated prior to sampling the voltage fluctuations caused by our proposed power-waster. These sensors are based on routing resources rather than delay line structures. Specifically, they exploit the timing sensitivity of long routed nets whose delays are influenced by local supply voltage fluctuations. A global clock signal is used as both the reference and the input signal to the sensor. This clock toggles the routed net, and output registers together with a counter record how many transitions arrive at the sensor output within a fixed sampling window. When the supply voltage drops, routing delays increase, resulting in fewer transitions being counted and hence lower sensor values. Conversely, higher voltages reduce routing delays, leading to higher toggle rates and sensor readings. In this experiment, both the PWNN and the sensors operate at 100 MHz, ensuring synchronized, cycle-accurate observation of supply voltage variations.

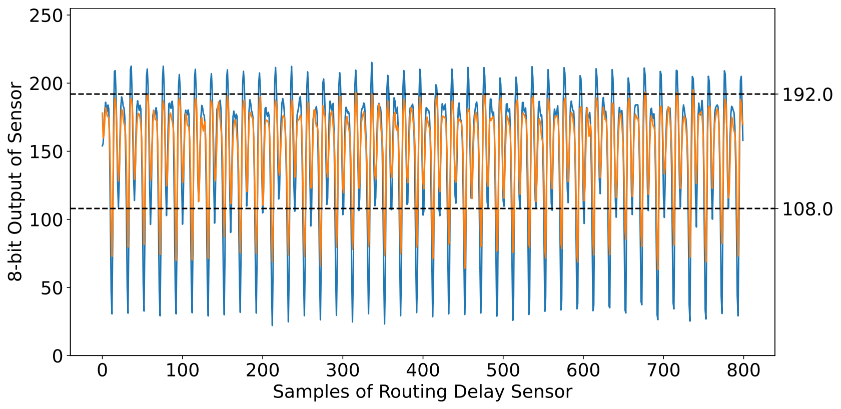


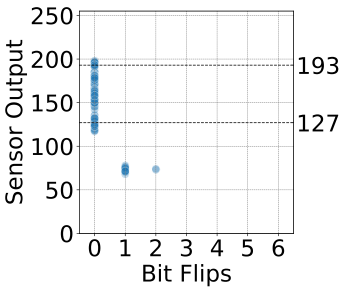
Figure 7: Power traces with 800 time points of 230×35 MLP as an example. The dashed line is the baseline with PWNN off, setting thresholds for voltage variations. The blue trace corresponds to PWNN running inference with a high power input, while the orange trace shows normal operation with random input.

The calibrated sensor initially runs with the PWNN deactivated while the collected power trace represents system noise in the idle state, which is labeled as the baseline condition. This baseline is used to define thresholds for identifying both voltage drops or voltage overshoots that may occur during recovery from such drops. The blue waveform corresponds to the PWNN executing with the optimized high power input, while the orange waveform reflects execution with random inputs as depicted in Figure 7. When the high power input is active, we observe significantly more pronounced voltage fluctuations, as captured by sharp dips in the sensor output. These fluctuations are consistent with large scale switching activity across the network and support our claim that the optimized input pattern triggers maximal dynamic power. In contrast, random inputs cause more moderate and uniform power variations. This visual distinction confirms the effectiveness of our proposed fault injection attack strategy in amplifying the likelihood of timing faults during sensitive circuit operations. Here, we use two optimized input images obtained during software based training, as described in Section 3. The input images are first stored in a custom buffer controller IP. This module provides fine-grained control over the accelerator's execution, allowing it to alternate between two inputs. Depending on the attack strategy, the controller can switch inputs either continuously for random fault injection or for a limited number of cycles to enable more precise, targeted faults.

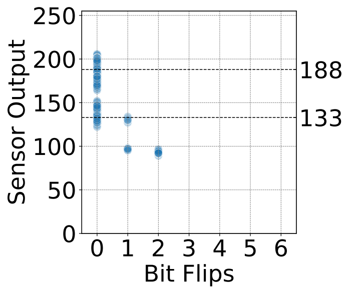
4.2.2 Fault Injection Attack Procedure and Result Analysis

To investigate how the depth of the adder tree affects fault intensity, we configure each XNOR bit operation to invert its output during training, thereby maximizing switching activity and making the XNOR-induced switching activity as uniform as possible across all six PWNN variants. In this way, we can clearly isolate and observe the influence of the depth of the adder tree on the fault sensitivity. In this configuration, all PWNNs complete the inference in exactly 10 cycles. In each attack iteration, the fault is applied to one specific clock cycle only, after which the RCA is reset in preparation for the next iteration.

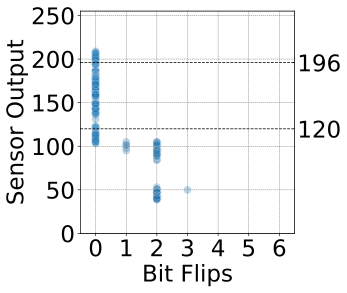
As illustrated in Figure 8, in order to observe the effect of voltage drop induced delay on carry propagation in the RCA, we apply two specific input patterns: one with all bits set (0xFFFF...FFFF) and another with a single least significant bit set (0x0000...0001). This configuration ensures full carry chain activation, thereby stressing the critical path. The x-axis in the figure denotes the number of bit transitions, which serves as an indicator of switching activity. An increased number of bit flips correlates with intensified voltage



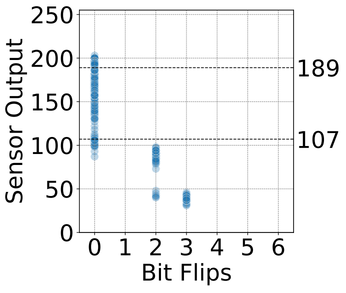
(a) 10×810 MLP: 3.00% NFR



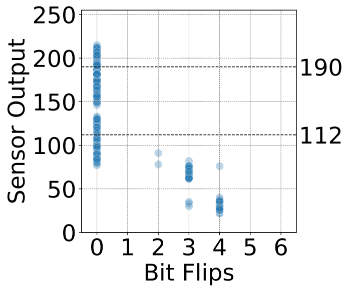
(b) 45×180 MLP: 5.48% NFR



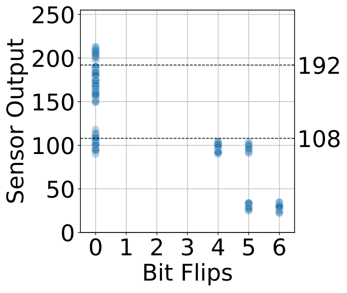
(c) 100x81 MLP: 6.50% NFR



(d) 150x54 MLP: 7.57% NFR



(e) 180x45 MLP: 11.50% NFR



(f) 230x35 MLP: 14.25% NFR

Figure 8: Fault intensity represented by NFR of 400 sample points across PWNNs with different adder tree depth.

drops induced by the PWNN, resulting in extended delays along the critical path.

The experimental results demonstrate the relationship between fault intensity and the corresponding power profile across different MLP configurations. For each experiment, faults were injected into the RCA during the clock cycle exhibiting peak switching activity, specifically cycles involving XNOR or accumulation operations. Concurrently, 400 voltage sensor samples were collected per configuration. The tested MLP architectures vary from compact networks (e.g., 10×810) to larger ones with more synapses (e.g., 230×35). Fault intensity is quantified as the Hamming distance (counted bit flips) between expected and faulty RCA outputs (x-axis), while sensor readings reflecting voltage drops via timing delays are plotted on the y-axis. To quantify the fault intensity in the PWNN output, we define Normalized Fault Rate (NFR) as:

$$\mathrm{NFR} = \frac{\sum_i \mathrm{HD}(O_i, O_i')}{N_{\text{fault}} \cdot Z_O}, \tag{5}$$

where $\mathrm{HD}(O_i, O_i')$ is the Hamming distance between the correct adder output O_i and the faulty output O_i', N_{fault} is the number of faulty samples (i.e., outputs where HD > 0), and Z_O is the number of zero bits in the correct output O_i. This normalization reflects the average number of bit flips per correct bit, conditioned on a fault occurrence.

A consistent trend emerges: networks with deeper adder trees and higher numbers of synapses have significantly higher fault intensity and lower sensor values (NFR of 14.25%), while smaller networks (10×810 and 45×180) have significantly lower rates of 3.00% and 5.72% respectively. This discrepancy emphasizes the crucial influence of the topology of the neural network, especially the depth of the adder tree, on the effectiveness of fault injection attacks with our PWNN approach. The increased cumulative switching activity and the longer carry propagation paths in deeper adder trees increase the fault susceptibility. Further analysis of the fault distribution shows that multi-bit faults (Hamming

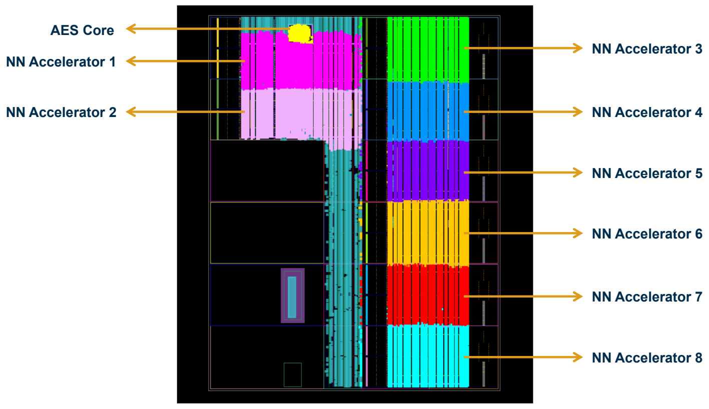


Figure 9: Post-placement floorplan on AMD ZCU104. The AES-128 core is located at the top of the device, and eight distributed instances of the 230×35 MLP are placed across the remaining area. At runtime, each PWNN instance can be independently activated using an 8-bit control mask (from LSB to MSB: NN1 to NN8), enabling dynamic control of switching activity and global placement.

distance ≥ 3) predominantly occur in deeper networks, while shallow configurations mostly exhibit single-bit faults, reinforcing the correlation between architecture depth and fault severity. Accordingly, power traces associated with higher fault intensities show pronounced amplitude reductions and often exceed detection thresholds with lower sensor values in deeper models. This comprehensive analysis confirms that the depth of the adder tree is a key factor for fault susceptibility in our attack model.

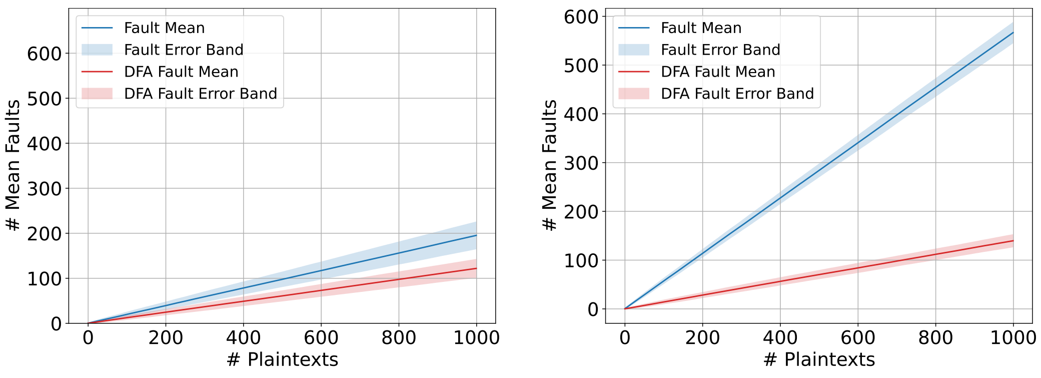
However, the depth of the adder tree cannot be increased indefinitely due to practical limitations imposed by the FINN framework and the FPGA architecture. In particular, the FINN framework requires additional resources to implement pipelining stages, which grow as the depth of the adder tree increases and affect the overall resource utilization. In addition, the FINN/Vitis HLS tool chain enforces a maximum limit of 8190 bits for the weighted input matrix, which limits the size of the MVAU layer and thus also the scalability of the network. Apart from resource utilization, deeper adder trees generate a larger number of control signals such as clock enables and resets, which require dedicated control sets on the FPGA. On AMD FPGA architectures, the number of these control sets is limited, and exceeding this budget can lead to routing congestion and implementation faults, even if the logical resources remain available. These combined factors limit the feasible depth of adder trees within the FINN framework and limit the scalability of the accelerator without more advanced architectural or tool chain optimizations.

4.3 DFA of AES-128 evaluated using PYNQ-Z2 and ZCU104

4.3.1 Dynamically Adjustable Multi-Instance PWNNs on ZCU104

Figure 9 illustrates the physical layout of our system on the AMD ZCU104 platform. The AES encryption core is located in the upper region, while eight neural network (NN) instances are distributed over the remaining area to maximize spatial coverage. Each NN implements a single-layer 230×35 MLP, which is the largest feasible configuration previously tested on the PYNQ-Z2 platform. On PYNQ-Z2, only a single instance could be deployed due to limited hardware and software constraints, as discussed in Subsubsection 4.2.2. In contrast, the larger resource capacity of ZCU104 enables the parallel use of several such instances.

The number of active PWNN instances can be set at runtime, while their placement



(a) PYNQ-Z2 **(b)** ZCU104

Figure 10: Fault rates observed over 500,000 AES encryptions with PWNNs active. On PYNQ-Z2 (1 instance), DFA-usable fault rate is 12.2%; on ZCU104 (5 instances under the best placement), DFA-usable fault rate is 13.9%.

is fixed at design time. After bitstream generation, instances are selectively enabled or disabled, with the attacker assumed to occupy a fixed FPGA region. Up to 8 PWNNs can run concurrently and their count and placement trade off fault effectiveness. In the next, we investigate how different number of NN accelerators and their dedicated global placement options affect the reliability of fault injection in our PWNN approach.

4.3.2 Evaluation of DFA Efficacy on PYNQ-Z2 and ZCU104

In this part, we evaluate our PWNN-based on-chip fault injection method across 500,000 AES-128 encryptions per device using 1000 uniformly random plaintexts. We record the cumulative mean of total faults and DFA-usable faults on both edge and cloud FPGA platforms as illustrated in Figure 10. On the PYNQ-Z2, a single PWNN instance achieves a total fault rate of 19.5% and a DFA-usable fault rate of 12.2%, demonstrating reliable injection within a narrow timing window but limited scalability due to resource and placement constraints. On the cloud FPGA Zynq UltraScale+ (ZCU104 development board), five PWNN instances as example are deployed under the corresponding best placement configuration. Despite the stronger PDN of the ZCU104 compared to PYNQ-Z2, the system achieves a significantly higher overall fault rate of 56.6% with a constant DFA-usable rate of 13.9%. These results show that scalable PWNN-based fault injection remains effective even under improved PDN conditions, enabling efficient DFA in cloud FPGA environments.

Figure 11 provides a placement-aware fault injection analysis on the ZCU104, showing the distribution of total and DFA-usable fault rates for 1 to 8 PWNN instances. For each instance count, we exhaustively evaluate all placement combinations by activating different PWNN instances in global placement as shown earlier in Figure 9, resulting in 256 placement combinations overall. This provides a comprehensive view of how spatial deployment impacts both scalability and fault injection effectiveness.

Total fault rate rises with PWNN count, but effectiveness is highly placement dependent for 1–3 instances: placing PWNNs near the AES core significantly boosts both total and DFA-usable faults, indicating strong spatial coupling between local voltage drop and AES timing. With 4–5 instances, placement sensitivity diminishes and fault rate is more uniformly, likely due to overlapping switching and a broader PDN disturbance. The DFA-usable rate is not monotonic, it increases at five instances and then decreases. At 7–8 instances, the total faults keep increasing but the DFA-usable rate decreases as noise and timing uncertainties from widely distributed PWNNs hinder synchronized

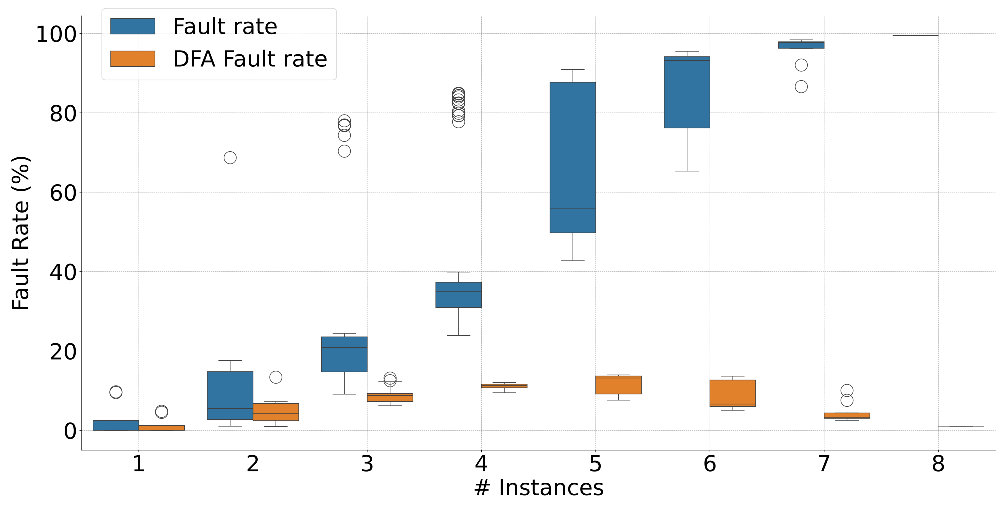


Figure 11: Analysis of total and DFA-usable fault rates on the ZCU104 across 500,000 AES-128 encryptions. For each PWNN instance count (1–8), all valid global placement combinations are evaluated. Results demonstrate the scalability of fault injection and the placement sensitivity of DFA-usable faults

drops (routing delays, clock skew, varying impedance), reducing the spatial precision and temporal alignment required for single-byte DFA faults.

Table 1 provides a quantitative summary of the results visualized in Figure 11, summarizing average, best and worst case of total and DFA fault rates for both ZCU104 and PYNQ-Z2 devices. The average total fault rate increases linearly from 2.8% at single PWNN instance to 99.5% at eight, indicating strong scalability of injected faults as more PWNNs are activated. The average DFA fault rate rises only up to five instances, reaching a best-case value of 13.9% before declining. Beyond this point, the fault activity becomes either too strong or too de-synchronized, making it difficult to consistently induce controlled, single-byte faults required for DFA. In the best case placement, both total and DFA fault rates vary non-monotonically across instance counts. Although other instance counts, such as 8 or 7 PWNNs, may have higher total fault rates approaching

Table 1: Average, best and worst fault rates (total and DFA-usable) on ZCU104 and PYNQ-Z2 according to placement pattern. For ZCU104, the best placement configuration is selected per instance count (For five active NN accelerators, the best placement enables instances 2, 4, 5, 6, and 7 as illustrated in Figure 9).

Device	**NN Instances**	**Total Fault Rate (%)**			**DFA Fault Rate (%)**		
		avg.	**best**	**worst**	**avg.**	**best**	**worst**
PYNQ-Z2	1	19.5	19.5	19.5	12.2	12.2	12.2
ZCU104	1	2.8	9.7	0.008	1.4	4.8	0.0
	2	10.2	68.7	1.0	4.6	13.4	0.9
	3	24.5	77.9	9.1	8.7	13.2	6.2
	4	43.4	84.9	23.8	11.1	12.1	9.4
	5	**65.2**	**90.9**	**42.7**	**11.8**	**13.9**	**7.6**
	6	84.4	95.5	65.3	9.2	13.7	5.0
	7	95.8	98.3	86.6	4.5	10.1	2.4
	8	99.5	99.5	99.5	1.1	1.1	1.1

Table 2: Required average number of plaintexts and corresponding execution time to obtain 8 correct/faulty ciphertext pairs for successful DFA over 1000 keys. Results are shown for the best placement configuration under instance counts, achieving optimal efficiency in extracting the required pairs for DFA.

Device	NN Instances	Plaintexts Used	Exec. Time (s)
PYNQ-Z2	1	248	2.85
ZCU104	1	1798	9.23
	2	134	0.69
	3	123	0.63
	4	131	0.67
	5	110	0.57
	6	**106**	**0.55**
	7	146	0.75
	8	1307	6.71

saturation, placement with 2, 3, 5, and 6 instances provides a better balance between fault quantity and precision. This emphasizes the importance of spatial distribution for effective single-byte fault injection.

Table 2 shows the average number of plaintexts and time required to collect sufficient correct/faulty ciphertext pairs for successful DFA across 1000 keys. Results are reported under the best placement configuration for each PWNN instance count on the ZCU104 and PYNQ-Z2. On PYNQ-Z2, a single PWNN instance requires 248 plaintexts and 2.85 seconds on average. On the ZCU104, activating five or six PWNN instances yields higher efficiency, requiring fewer than 110 plaintexts and reducing collection time to under 0.6 seconds. However, best placement with only one or as many as eight instances requires significantly more plaintexts and time, making them less practical for real-world DFA.

In general, activating more PWNN instances increases switching activity but also introduces synchronization challenges that reduce fault injection precision. With limited resources, as in cloud FPGA scenarios, placing the adversary close to the victim core is most effective, especially with 1–3 instances. At 5 instances, however, switching activity is strong enough that proximity is no longer required, making this configuration the most effective overall.

5 Discussion

In this work, we demonstrated that malicious neural networks can act as power-wasting circuits. The key insight is that those binary NNs do not require any significant modifications in their usual topology, which effectively makes them indistinguishable from benign.

As we have shown, even when a neural network is generated using an established open-source framework, FINN, it can be configured and trained to maximize switching activity during operation, thereby increasing dynamic power consumption and causing voltage fluctuations that can be exploited in fault injection attacks without further modification. We believe that a NN generated and unmodified in this way, decoupled from the weights, is already practically indistinguishable from benign. Moreover, the weights can usually be reconfigured at runtime, which provides the ability to execute some proper workload. In this way, a potential adversary can switch between benign and attack modes by loading different sets of weights, using the benign mode as additional camouflage. Moreover, it is very difficult to predict malicious intentions in advance if the input patterns that activate the power-wasting properties are unknown and depend on training.

Therefore, we believe that both DRC and other formal and analytical methods are rather unable to detect malicious networks with acceptable probability or lead to a high number of false positives, which would make these detection approaches practically infeasible. Furthermore, the proposed method demonstrates good scalability in cloud FPGA platform, although its effectiveness may be influenced by placement and routing constraints such as component proximity and routing paths. In addition, the exact architecture and size of the PWNN can be slightly modified each time to avoid possible detection by signature/pattern scanning. This will also likely help to avoid detection by machine learning techniques applied to bitstreams or netlists. Overall, we believe that the detection of such malicious neural networks deployed in cloud computing environments is practically infeasible.

Techniques such as continuous runtime monitoring of voltage/delay fluctuations with anomaly detection, as well as error detection and error correction, would be suitable to detect and counter an ongoing attack. However, it still probably would require a lot of manual effort to pinpoint the exact source of the voltage fluctuations, since the PWNN is considered to execute an attack for a very short time in total.

Further, it should be possible to trade off some stealthiness and craft fully custom PWNNs with more significant changes to the usual topology and activation functions, which will result in even more switching activity and dynamic power consumption, making it suitable for more intense voltage drops. Additionally, the intensity of the voltage drops may be controlled almost gradually by the changes to the activation input patterns. So, some sort of Fault Sensitivity Analysis (FSA) attacks may also be possible where only knowledge if the encryption was faulty or fault-free is necessary and it is not required to encrypt the same plaintext at least two times or know the exact outputs.

6 Conclusion

We introduced PWNN, a stealth neural accelerator that induces targeted voltage fluctuations by maximizing switching activity in binarized neural networks. Unlike conventional power wasters, PWNN circuits are synthesized using standard tool flows, pass all design rule checks, and embed malicious behavior only in their inputs and trained weights.

Our experiments show that PWNNs can reliably induce timing faults in victim AES cores on both resource-constrained and commonly used cloud FPGA platforms. On cloud FPGA Zynq UltraScale+, we are able to dynamically adjust the number of active PWNN instances and selectively control their placement within the global FPGA fabric at runtime. This flexibility enables effective generation of DFA-usable faults and rapid recovery of the required correct/faulty ciphertext pairs. These results demonstrate that functionally correct BNN accelerators, when adversarially trained, can act as effective fault injectors by inducing power-induced timing violations in nearby logic.

On the other hand, PWNN represents a new attack paradigm that challenges the assumptions underlying FPGA isolation and bitstream security. In the future, the defense needs to evolve from static validation to runtime monitoring of activity profiles, especially in shared computing environments. Our work motivates further research on adversary-trained hardware accelerators and their impact on the security of reconfigurable platforms.

Acknowledgments

This work is funded by the German Federal Ministry of Research, Technology and Space (BMFTR) in the framework of DE:Sign DI-EDAI (grant no. 16ME0990K-16ME0993). We are grateful for the anonymous reviewers' efforts and helpful recommendations.

References

[Ama25] Amazon Web Services. Amazon EC2 F1/F2 Instances. https://aws.amazon.com/ec2/instance-types/f2/, 2025.

[AMD24] AMD. Introducing amd cdna™ 3 architecture. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf, 2024.

[AMD25] AMD. Amd instinct™ mi300a apu data sheet. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/data-sheets/amd-instinct-mi300a-data-sheet.pdf, 2025.

[ANK+24] Lilas Alrahis, Hassan Nassar, Jonas Krautter, Dennis Gnad, Lars Bauer, Jörg Henkel, and Mehdi Tahoori. Malignnoma: Gnn-based malicious circuit classifier for secure cloud fpgas. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 383–393. IEEE, 2024.

[CC22] J. Chaudhuri and K. Chakrabarty. Detection of Malicious FPGA Bitstreams Using CNN-Based Learning. In *IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2022.

[CNG+24] Jayeeta Chaudhuri, Hassan Nassar, Dennis R.E. Gnad, Jörg Henkel, Mehdi B. Tahoori, and Krishnendu Chakrabarty. Hacking the fabric: Targeting partial reconfiguration for fault injection in fpga fabrics. In *2024 IEEE 33rd Asian Test Symposium (ATS)*, pages 1–6, 2024.

[CZL+25] Shixin Chen, Hengyuan Zhang, Zichao Ling, Jianwang Zhai, and Bei Yu. The survey of chiplet-based integrated architecture: An eda perspective. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASPDAC '25)*, pages 1–10, Tokyo, Japan, 2025.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on aes. In *International Conference on Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.

[DSZ21] Ghada Dessouky, Ahmad-Reza Sadeghi, and Shaza Zeitouni. Sok: Secure fpga multi-tenancy in the cloud: Challenges and opportunities. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 487–506. IEEE, 2021.

[EKC19] Rana Elnaggar, Ramesh Karri, and Krishnendu Chakrabarty. Multi-tenant fpga-based reconfigurable systems: Attacks and defenses. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 7–12, 2019.

[FAFK+24] Othmane Friha, Mohamed Amine Ferrag, Burak Kantarci, Burak Cakmak, Arda Ozgun, and Nassira Ghoualmi-Zine. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. *IEEE Open Journal of the Communications Society*, 5:5799–5856, 2024.

[GOKT18] Dennis RE Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B Tahoori. An experimental evaluation and analysis of transient voltage fluctuations in fpgas. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(10):1817–1830, 2018.

[GOT17] Dennis RE Gnad, Fabian Oboril, and Mehdi B Tahoori. Voltage drop-based fault attacks on fpgas using valid bitstreams. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[HBH24] Hossam A. Helaly, Mahmoud Badawy, and Ahmed Y. Haikal. A review of deep learning approaches in clinical and healthcare systems based on medical image analysis. *Multimedia Tools and Applications*, 83:36039–36080, 2024.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KGT18] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. Fpgahammer: Remote voltage fault attacks on shared fpgas, suitable for dfa on aes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 44–68, 2018.

[KGT19] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. Mitigating electrical-level attacks towards secure multi-tenant fpgas in the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(3):1–26, 2019.

[KGT21] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. Remote and stealthy fault attacks on virtualized fpgas. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1632–1637. IEEE, 2021.

[KSY23] Mashrafi Alam Kajol, Sandeep Sunkavilli, and Qiaoyan Yu. Ahd-lam: A new mitigation method against voltage-drop attacks in multi-tenant fpgas. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, 2023.

[LHWW22] Zeyu Li, Zhao Huang, Junjie Wang, and Quan Wang. Investigate of mitigation solution against hardware trojans attack on evolvable hardware platform. In *2022 19th International SoC Design Conference (ISOCC)*, pages 213–214, 2022.

[LMG$^+$20] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale+ fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 13(3):1–31, 2020.

[LPPK23] Tuan La, Khoa Pham, Joseph Powell, and Dirk Koch. Denial-of-service on fpga-based cloud infrastructures: Attack and defense. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023.

[Man25] Murali Krishna Reddy Mandalapu. Emerging chiplet-based architectures for heterogeneous integration. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 11(2):1081–1098, 2025.

[MDH$^+$22] Dina G Mahmoud, David Dervishi, Samah Hussein, Vincent Lenders, and Mirjana Stojilović. Dfaulted: Analyzing and exploiting cpu software faults caused by fpga-driven undervolting attacks. *IEEE Access*, 10:134199–134216, 2022.

[Mic25] Microsoft Azure. Microsoft azure. https://azure.microsoft.com/, 2025.

[MLPK20] Kaspar Matas, Tuan Minh La, Khoa Dang Pham, and Dirk Koch. Powerhammering through glitch amplification – attacks and mitigation. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–69, 2020.

[MSRZ+25] Amit Mazumder Shuvo, Md Latifur Rahman, Jingbo Zhou, Farimah Farahmandi, and Mark Tehranipoor. Refid: A system-aware remote fault-injection attack detection & mitigation for secure heterogeneous system. In *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 46–56, 2025.

[MZTF22] Md Rafid Muttaki, Tao Zhang, Mark Tehranipoor, and Farimah Farahmandi. Ftc: A universal sensor for fault injection attack detection. In *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 117–120. IEEE, 2022.

[Nat23] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Technical Report FIPS PUB 197-upd1, U.S. Department of Commerce, May 2023. Originally published November 26, 2001; updated May 9, 2023.

[OLK25] Murat Arda Onsu, Poonam Lohan, and Burak Kantarci. Leveraging edge intelligence and llms to advance 6g-enabled internet of automated defense vehicles. *IEEE Internet of Things Magazine*, 8(4):148–155, 2025.

[PHT20] George Provelengios, Daniel Holcomb, and Russell Tessier. Power wasting circuits for cloud fpga attacks. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 231–235. IEEE, 2020.

[SGS23] Dominik Spielmann, Ognjen Glamočanin, and Milan Stojilović. RDS: FPGA routing delay sensors for effective remote power analysis attacks. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, volume 2023, pages 543–567. Ruhr University Bochum, 2023.

[SZY21] Sandeep Sunkavilli, Zhiming Zhang, and Qiaoyan Yu. New security threats on fpgas: From fpga design tools perspective. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 278–283, 2021.

[TCC+24] Sumit Tanwar, Nipun Choudhary, Vaibhav Chaudhary, Anish Dahiya, Priyanka Kaushik, and Rachna Rathore. Advancing healthcare: Cnn-based brain hemorrhage detection in intelligent environments. In *2023 International Conference on Smart Devices (ICSD)*, pages 1–6, 2024.

[UJFG+17] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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 (FPGA)*, pages 65–74. ACM, 2017.

[Xil19] Xilinx. Brevitas: Quantization-aware training in pytorch. https://github.com/Xilinx/brevitas, 2019.

[YWC+24] Chenxi Yang, Xinyu Wang, Peng Chen, Zhihui Lu, and Xiaozheng Du. Finsec: An efficient microservices-based detection framework for financial ai model security. In *2024 IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 101–106, 2024.

[ZGK22] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. Shef: Shielded enclaves for cloud fpgas. In *Proc. ACM ASPLOS*, 2022.

[ZS25] Mark Zhao and G. Edward Suh. Remote power side- channel attacks on fpgas. *IEEE Design & Test*, 42(1):13–19, 2025.

[ZVF$^+$21] Shaza Zeitouni, Jo Vliegen, Tommaso Frassetto, Dirk Koch, Ahmad-Reza Sadeghi, and Nele Mentens. Trusted configuration in cloud fpgas. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 233–241. IEEE, 2021.

[ZWY$^+$24] Jinming Zhang, Xuyan Wang, Yaoyao Ye, Dongxu Lyu, Guojie Xiong, Ningyi Xu, Yong Lian, and Guanghui He. M2m: A fine-grained mapping framework to accelerate multiple dnns on a multi-chiplet architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(10):1864–1877, 2024.

[ZYCH22] Zheng Zhuang, Bei Yu, Kuan-Yu Chao, and Tsung-Yi Ho. Multi-package co-design for chiplet integration. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9, 2022.