Shuffled-ArgMax: Securing ArgMax Decisions Against Side-Channel-Guided Fault Injection in Edge-AI IoT Devices

Le Wu[†], Liji Wu[†*], Yuyang Pan[‡], Xiangmin Zhang[†], and Jian Wu[§]
[†]School of Integrated Circuits, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China
[‡] Beijing UnionPay Card Technology Co., Ltd, Beijing, China
[§] Beijing Pairui Micro Technology Co., Ltd, Beijing, China
[†]wul22@mails.tsinghua.edu.cn, [†]{lijiwu, zhxm}@mail.tsinghua.edu.cn, [‡]panyuyang@bctest.com, [§]wujian@ahcx.cloud
[*]Corresponding author

***Abstract*—Edge-AI Internet-of-Things (IoT) devices increasingly perform local neural-network inference in physically accessible environments. While prior physical attacks on neural networks mainly focus on model extraction, parameter recovery, or intermediate computation, the final decision stage remains insufficiently studied. This paper investigates the physical security of ArgMax-based decision logic, which converts output scores into the final class label in many edge-AI deployments. We show that a sequential ArgMax implementation can expose deterministic power-trace patterns associated with loop iterations and conditional maximum updates. These patterns can be used to recover decision-related timing information and guide voltage fault injection, enabling targeted redirection of the final inference result. We implement the attack on a real MCU-based edge-AI platform using an STM32F407VG device running an X-CUBE-AI-deployed convolutional neural network. Under dual-point voltage fault injection, the unprotected ArgMax achieves targeted redirection to all non-original classes in the MNIST case study. To mitigate this threat, we propose `Shuffled-ArgMax`, a lightweight software-level defense that combines randomized traversal, loop-integrity checking, and redundant decision verification. Experimental results show that, under the same attack setting, `Shuffled-ArgMax` reduces the targeted redirection success rate from 56.92% to 0%, demonstrating strong resilience against targeted misclassification attacks. Meanwhile, it introduces only a small end-to-end runtime overhead relative to the complete DNN inference.**

***Index Terms*—Edge-AI IoT security, ArgMax, side-channel analysis, voltage fault injection, targeted misclassification, lightweight countermeasure.**

I. Introduction

With the widespread deployment of Artificial Intelligence (AI) technologies in Internet-of-Things (IoT) and embedded edge devices, Deep Neural Networks (DNNs) are increasingly executed locally on resource-constrained platforms. These edge-AI devices are often deployed in physically accessible or semi-controlled environments, such as sensing nodes, access-control terminals, industrial inspection devices, and autonomous monitoring systems. Unlike cloud-based inference, where physical access to the computing infrastructure is typically restricted, edge-AI hardware may be directly exposed to local adversaries. This makes physical attacks a practical threat to the confidentiality and integrity of on-device DNN inference.

Among these threats, Side-Channel Attacks (SCAs) and Fault Injection Attacks (FIAs) are particularly relevant to edge-AI IoT devices. SCAs exploit physical leakages such as power consumption, electromagnetic emanation, or timing behavior to infer sensitive information without modifying the algorithmic logic or damaging the device [1]. Recent studies have shown that side-channel analysis can reveal model parameters [2]–[4], network architectures [4]–[6], and even input data [7]–[9] from neural-network implementations. These results indicate that DNN inference may unintentionally expose substantial information through physical channels. However, most existing SCA studies focus on model extraction, architecture recovery, or input reconstruction, while the final decision output of DNN inference has received much less attention.

FIAs, on the other hand, actively disturb the normal execution of DNN hardware by injecting transient physical perturbations. Such attacks may induce random misclassification or force the inference result toward attacker-favored labels. Existing studies have demonstrated the feasibility of targeted fault attacks on neural-network inference, but several limitations remain. First, many FIA techniques either cause random misclassification or force outputs into a fixed or limited range of classes [10], [11]. Second, some attacks rely on strong adversarial capabilities, such as direct access to output feedback, I/O interfaces, runtime control, or profiling information [10]–[12]. Third, although recent work has begun to investigate fault injection on decision-related units such as ArgMax, the security implications of the final decision stage under combined side-channel observation and fault injection remain insufficiently explored [12].

This paper focuses on the ArgMax-based final decision logic in DNN inference. Although ArgMax is lightweight compared with convolution, matrix multiplication, or other major computational layers, it directly converts output scores into the final predicted class and therefore forms a critical decision boundary in edge-AI IoT systems. We show that a typical sequential ArgMax implementation exhibits deterministic loop traversal, repetitive comparison patterns, and data-dependent maximum updates, which can be observed through power side channels. By using these side-channel features to guide voltage fault injection, an adversary can localize critical decision

TABLE I
COMPARISON WITH PRIOR TARGETED MISCLASSIFICATION ATTACKS AND COUNTERMEASURES

Work	Misclassification Goal	Fault Calibration	Glitch Type	Platform	Dataset	Attack Target	Countermeasure
[10]	Targeted range of classes	Output feedback	Clock	AMD KV260	ImageNet	FC layers	None
[11]	Fixed-class redirection	Waveform matching	Clock	ATXmega128	MNIST	Softmax	None
[12]	Targeted class skipping	Profiling	Clock	XC7A100T	MNIST	ArgMax	None
This Work	**Targeted class redirection**	**SCA-guided output recovery**	**Dual-point voltage**	**STM32F407VG**	**MNIST**	**ArgMax**	**Shuffled-ArgMax**

events and redirect the final prediction to an adversary-chosen class.

Rather than introducing a new physical attack primitive, this work reveals an overlooked decision-stage attack surface in edge-AI IoT devices. We present a side-channel-guided fault injection methodology for ArgMax-based decision logic and further explore a lightweight software-level countermeasure, `Shuffled-ArgMax`, to disrupt trace alignment and improve decision integrity. The main contributions of this work are summarized as follows:

- We identify ArgMax-based final decision logic as an overlooked physical attack surface in edge-AI IoT devices, where corrupting the decision stage can directly compromise inference integrity.
- We present a side-channel-guided dual-point voltage fault injection methodology that exploits power-trace patterns of sequential ArgMax execution to localize decision events and enable targeted output redirection.
- We implement and evaluate the proposed attack on a real STM32F407VG-based edge-AI platform running an X-CUBE-AI-deployed CNN, and quantify success, failure, crash, and defense-triggered outcomes.
- We propose `Shuffled-ArgMax`, a lightweight software-level defense that combines traversal randomization, loop-integrity checking, and redundant decision verification to disrupt trace-aligned fault injection.

II. BACKGROUND

A. Combined Attacks and Countermeasures

1) Combined Side-Channel and Fault Injection Attacks: SCAs exploit physical leakages generated during algorithm execution. In microcontrollers (MCUs), the instantaneous power consumption of CMOS circuits can create distinguishable patterns in recorded power traces. Simple power analysis (SPA), one of the classical SCA techniques, analyzes these patterns to infer internal operations and recover sensitive information. Prior studies have demonstrated that power traces can be used to recover input images [7], [9], [13], extract model architectures [4], [6], and steal model parameters such as weights [4], [14], [15]. Despite extensive prior work on DNN side-channel leakage, the leakage and manipulation of final decision logic, especially ArgMax-based decision stages, remain insufficiently explored.

FIAs actively disturb the physical environment of embedded devices to disrupt normal computation. Techniques such as

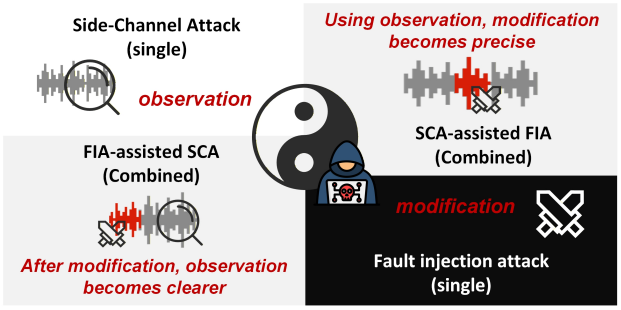


Fig. 1. Two categories of combined side-channel and fault injection attacks.

voltage glitching [16], clock glitching [17], electromagnetic pulses [18], and laser injection [19] can induce control-flow deviations, register bit flips, or memory corruption. While SCAs and FIAs are often studied independently, they can also complement each other in combined attacks. As illustrated in Fig. 1, such attacks can generally be categorized into two classes: (i) fault-assisted side-channel attacks, where injected faults perturb device execution and amplify side-channel leakage [19]; and (ii) side-channel-guided fault attacks, where side-channel observations are used to localize effective fault-injection timing or fault windows [10], [11], [20].

2) Countermeasures Against Combined Attacks: Countermeasures against physical attacks on DNN inference have been explored from both side-channel and fault-injection perspectives. In side-channel security, shuffling is a widely adopted hiding countermeasure. Its core idea is to randomize the execution order of sensitive operations, thereby preventing adversaries from constructing stable leakage models from power or electromagnetic traces [21]. In neural-network implementations, shuffling has been applied to protect certain arithmetic operations, such as multiplication or accumulation steps [22], [23]. However, existing shuffling-based defenses mainly target computational layers or arithmetic operations, and have not been tailored to side-channel-guided fault injection against ArgMax-based final decision logic.

Compared with SCA countermeasures, defenses against FIAs on DNN hardware remain relatively limited and are often empirical. Existing approaches mainly include redundant computation, consistency checking, fault detection, and execution randomization. For example, Dumont *et al.* incorporated bit checking and redundant computation in parameter storage to detect laser-induced parameter corruption [19]. Lee *et al.* introduced a clock anomaly detection mechanism to capture glitch-induced faults during the execution of Softmax and

Sigmoid functions [20]. Despite these efforts, countermeasures designed for combined side-channel and fault-injection attacks remain scarce, particularly for protecting the final ArgMax decision stage. A suitable defense should not only reduce side-channel observability, but also disrupt fault-window localization and prevent localized faults from being converted into controlled output redirection.

B. The ArgMax Unit

During DNN inference, the model typically outputs a probability distribution vector over all classes, denoted as $\mathbf{p} = (p_1, p_2, \ldots, p_K)$, where K represents the number of classes. The final predicted class $\hat{y}$ is usually determined by the ArgMax operation, which is mathematically defined as:

$$\hat{y} = \arg \max_{i \in \{1,2,\ldots,K\}} p_i, \tag{1}$$

In classification-based DNN inference, the ArgMax function acts as the final post-processing and decision unit, selecting the class with the highest predicted probability as the model's output. Although it does not participate in the training process and is relatively lightweight compared with major computational blocks such as convolution engines or Multiply Accumulate (MAC) arrays, it is critical at the deployment and application levels because it converts prediction scores into actionable decisions and directly determines the final inference result. Therefore, despite its simple functionality and modest hardware footprint, the ArgMax unit represents a security-sensitive component in the inference pipeline. However, its security implications under combined side-channel and fault injection attacks remain largely unexplored.

C. Related Work

Recent studies have explored class- and output-related leakage in neural-network inference. Shukla *et al.* introduced Class-Leakage and showed that non-constant-time branching in PyTorch MaxPool can leak input class labels through timing channels [24]. Coqueret *et al.* demonstrated that electromagnetic leakage from the Softmax computation can be used to extract logits, thereby enabling black-box adversarial-example generation [25]. ScaAR further showed that electromagnetic side channels from real-world edge neural-network implementations can reveal user-interaction labels, including input and output class labels [26]. These works confirm that class- and output-related information can leak through physical or timing side channels. However, these attacks are mostly profiling-based. They typically require a manifest dataset, an open-device profiling phase, or a large number of labeled traces/timing samples to train a trace-to-label or trace-to-logit mapping. Moreover, they mainly focus on passive information recovery from MaxPool, Softmax, or implementation-level trace features. To the best of our knowledge, the power-leakage patterns of the ArgMax function itself, especially its deterministic traversal and maximum-update behavior, have not been systematically studied for interpretable decision or label recovery. More importantly, prior output-recovery studies do not further investigate how such decision-related leakage can be used to guide fault injection for targeted misclassification.

Misclassification is one of the most critical attack objectives and can be divided into non-targeted and targeted types. Non-targeted attacks merely degrade the overall inference accuracy, whereas targeted attacks redirect predictions to an adversary-chosen class and therefore pose a stronger threat. Several prior efforts have explored targeted misclassification, as summarized in Table I. Ordoñez *et al.* achieved class-range redirection on FPGA devices, but their attack cannot force an arbitrary specific class [10]. Fukuda *et al.* showed that clock glitching can force predictions into a fixed class, offering no fine-grained output controllability [11]. The skip-class attack bypasses a target class, but still cannot dictate the final output and requires a costly profiling phase to construct the fault window [12]. Overall, prior targeted-misclassification attacks typically rely on direct output observability, explicit profiling, or other auxiliary information to calibrate effective fault windows, and they provide limited control over the final output. In contrast, this work targets the ArgMax final decision stage and uses power side-channel leakage as an auxiliary observation channel for fault injection. Rather than relying on direct DUT I/O to localize the decision stage, the adversary infers decision-related execution features from the ArgMax power trace and uses them to guide voltage fault injection for adversary-chosen output redirection.

Countermeasures for ArgMax-related side-channel leakage have also begun to emerge. ARMOR [27] proposes first-order masked activation and ArgMax gadgets for side-channel-resistant neural networks. Its masked ArgMax design is built upon secure comparison and sign-extraction gadgets, aiming to protect trained model parameters, such as weights and biases, against first-order power analysis and probing attacks. However, ARMOR focuses on passive SCA/probing-based model stealing and does not address side-channel-guided fault injection or targeted decision manipulation. Different from masking-based ArgMax protection, our `Shuffled-ArgMax` defense randomizes the traversal order of the final decision logic and combines it with loop-integrity checking and redundant decision verification. Its goal is to disrupt trace alignment and fault-window localization, thereby mitigating targeted output redirection at the ArgMax stage.

III. Threat Model

We consider a local physical adversary targeting an edge-AI IoT device deployed in an open or semi-controlled environment. The adversary has physical access to the device package or power supply path, but cannot modify the firmware or directly read internal states, intermediate outputs, model parameters, or runtime variables. The adversary does not know the trained model parameters or the output vector values. However, we assume a limited gray-box setting in which the adversary does not know the model parameters or output values, but knows or can infer that the deployed inference firmware uses a sequential ArgMax-like post-processing routine to convert the output probability vector into the final predicted label. This assumption is realistic for many embedded inference libraries, where the final decision routine is simple and commonly reused across models.

The attack proceeds in two stages. In the first stage, the adversary performs non-profiled power side-channel analysis during ArgMax execution. By collecting power traces of the final decision stage, the adversary aims to identify deterministic execution patterns, such as loop iterations, comparison operations, and maximum-update events. These features can reveal decision-related timing information and provide timing anchors for subsequent fault injection.

In the second stage, the adversary actively injects voltage glitches during ArgMax execution. Guided by the side-channel-observed timing features, the adversary localizes critical fault windows and injects transient voltage disturbances to corrupt the normal comparison, branch, or loop-control behavior of the ArgMax routine. The attack objective is targeted misclassification: forcing the final predicted label to an adversary-chosen class rather than merely causing random inference failure.

In summary, we assume an adversary with the following capabilities:

- The adversary has local physical access to the target edge-AI device and its power supply path.
- The adversary can collect power traces during inference, especially around the ArgMax execution window.
- The adversary can inject voltage glitches with fine-grained timing control.
- The adversary has no privileged access to firmware, internal states, intermediate outputs, model parameters, or runtime variables.
- The adversary does not know the trained model parameters or output vector values, but knows that a sequential ArgMax-like final decision routine is used.

This threat model is relevant to physically exposed edge-AI IoT applications, where corruption of the final decision may cause harmful system-level consequences. Examples include local access control, industrial inspection, and sensor-edge intelligence, in which a wrong decision may lead to unauthorized access, defective products being accepted as normal, or erroneous outputs being propagated to higher-level control logic. Therefore, the final decision stage should be treated as a meaningful security boundary rather than a benign post-processing step.

IV. Hardware Security Analysis on ArgMax

A. Overall Attack Strategy

This study focuses on DNN inference deployed on embedded edge-AI IoT devices. As illustrated in Fig. 2, the host computer communicates with the target device through a UART interface to transmit input images and trigger the inference process. A typical embedded DNN inference pipeline consists of input preprocessing, forward propagation, output-score computation, and final post-processing. The final post-processing step is usually implemented by the ArgMax unit, which compares the output scores and selects the class with the highest value as the predicted label. Although this step is lightweight, it directly determines the final inference result. Therefore, corrupting the ArgMax decision logic can lead to

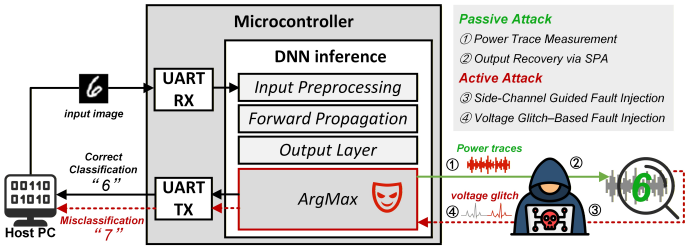


Fig. 2. Overview of the proposed side-channel-guided fault injection attack on the ArgMax unit during DNN inference.

targeted misclassification, where the output is redirected to an adversary-chosen class.

To achieve targeted redirection, we propose a side-channel-guided fault injection strategy. The attack first collects power traces during ArgMax execution and applies SPA to identify decision-related execution patterns. In a sequential ArgMax implementation, loop iterations, comparison operations, and maximum-update events may produce distinguishable power features. These features allow the adversary to infer the intermediate decision process and localize candidate fault windows. The adversary then injects voltage glitches at selected time points to disturb the comparison, branch, or loop-control behavior of the ArgMax routine, thereby altering the final predicted label.

Fig. 2 illustrates the basic attack flow. Under normal execution, the embedded device correctly classifies the input image received from the host, e.g., recognizing the handwritten digit "6". During the attack, the adversary analyzes the ArgMax power trace to identify timing anchors associated with the decision process. A voltage glitch is then injected during the critical decision window, disrupting the ArgMax computation and redirecting the final output to a target class, e.g., "7". This shows that the final decision stage can become an exploitable security boundary in embedded Edge-AI IoT systems.

B. Output Recovery Attack on ArgMax

Before presenting the SPA-based recovery method, we first analyze the attack surfaces of a sequential ArgMax implementation on embedded IoT devices. As described in Section II-B, a standard ArgMax iteratively scans the output probability vector, compares each element with the current maximum, and updates both the maximum value and its index when the comparison condition is satisfied, as shown in Alg. 1. This simple control flow creates three exploitable side-channel surfaces.

a.1 Deterministic traversal: The output vector is scanned in a fixed order, enabling a stable mapping between loop iterations and class indices.
a.2 Repetitive loop patterns: Each iteration executes similar comparison logic, producing recurring leakage patterns in the power trace.
a.3 Conditional maximum update: When a new maximum is found, the branch and update operations introduce data-dependent execution features.

Algorithm 1 Sequential ArgMax Decision Procedure

Require: A probability distribution vector $\mathbf{p}$ over K classes
Ensure: Index of the maximum value

```
max_idx ← 0
max_val ← p[0]
for i ← 1 to K − 1 do
    if p[i] > max_val then
        max_val ← p[i]
        max_idx ← i
    end if
end for
return max_idx
```

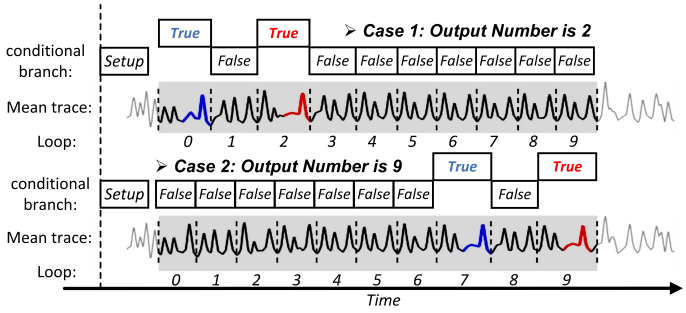


Fig. 3. SPA-based recovery of ArgMax outputs from power traces.

Based on these observations, we develop an SPA method targeting the ArgMax unit. As shown in Fig. 3, the power trace contains periodic patterns caused by the repeated comparison logic in the sequential ArgMax routine. The adversary first detects these recurring peaks and uses them as timing anchors to segment the trace into individual loop-iteration windows, corresponding to attack surface a.2. Each window represents one traversal step over the output vector and therefore contains the leakage of one comparison between the current score and the stored maximum.

After the iteration windows are identified, the adversary exploits the deterministic traversal order of ArgMax, corresponding to attack surface a.1. Since a standard sequential ArgMax scans the output vector from the first class to the last class in a fixed order, the segmented windows can be mapped to class indices according to their temporal order. For example, the first detected iteration window corresponds to the first visited class, the second window corresponds to the next class, and so on. This mapping allows the adversary to associate the observed power features with specific output indices without requiring access to the internal score values. It further provides the basis for recovering the maximum-update trajectory in the following step.

The conditional maximum update provides another important leakage feature, corresponding to attack surface a.3. When the current score exceeds the stored maximum, the program executes additional branch and update operations to refresh max_val and max_idx. These update events introduce distinguishable power and timing features, marked by the blue and red regions in Fig. 3.

After segmenting the trace into iteration windows and mapping them to class indices using attack surfaces a.1 and a.2, the adversary identifies which windows contain maximum-update events based on attack surface a.3. Each detected update means that max_idx is replaced by the class index of the current window. By tracking these updates along the deterministic traversal order, the adversary can reconstruct the evolution of max_idx and recover the final predicted label. In particular, the last successful update, highlighted by the red waveform in Fig. 3, determines the global maximum index and is therefore critical for correct output recovery.

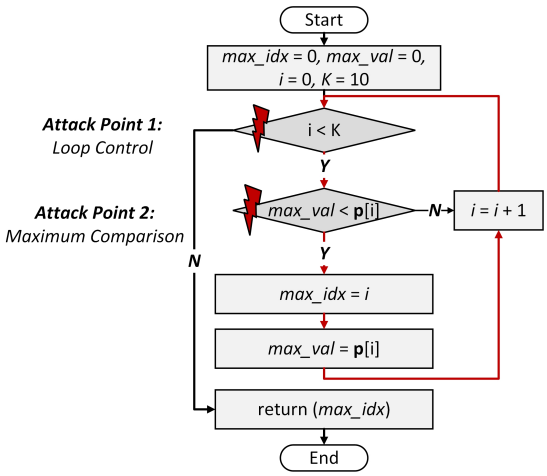


Fig. 4. ArgMax control flow and potential fault injection points enabling targeted misclassification.

C. Targeted Misclassification on ArgMax

The output recovery analysis in Section IV-B reveals not only the final prediction, but also decision-related timing information, including loop iterations and conditional update events. Such information can significantly facilitate fault injection attacks. Based on the sequential ArgMax procedure in Alg. 1 and its control flow in Fig. 4, we identify two fault-sensitive attack surfaces that can be exploited for targeted misclassification.

b.1 Loop-control faults: Faults in the loop-control logic may prematurely terminate the ArgMax traversal, causing the routine to return an intermediate maximum index.
b.2 Maximum comparison/update faults: Faults during comparison or conditional update may corrupt the branch outcome and incorrectly update max_val or max_idx.

In practice, these two attack surfaces correspond to two concrete fault injection points.

Attack Point 1: Loop Control. This point corresponds to the loop-termination condition. A voltage glitch here prematurely ends the loop, causing the algorithm to return the local maximum observed before the injection, thereby compromising ArgMax integrity. Notably, if the adversary injects a fault only at Attack Point 1 (e.g., aiming to redirect the result to index t), the attack will succeed only if $p[t]$ indeed happens to be the maximum among the first t elements. In such cases, the final comparison naturally triggers the update and the loop

Algorithm 2 Dual-Point Fault Injection Against ArgMax

Require: ArgMax power trace $\mathcal{T}$, target index t, number of classes K
Ensure: Attack outcome and final predicted label $\hat{y}$
Detect the ArgMax execution region in $\mathcal{T}$
Segment $\mathcal{T}$ into K loop-iteration windows $\{W_0, W_1, \ldots, W_{K-1}\}$
Select the target iteration window W_t corresponding to target index t
Locate Attack Point 2 in W_t, i.e., the comparison/update window
Locate Attack Point 1 after the comparison/update operation in W_t, i.e., the subsequent loop-control window
Configure glitch g_2 at Attack Point 2 to corrupt the comparison/update behavior
Configure glitch g_1 at Attack Point 1 to terminate the traversal before later overwrites
Run DNN inference and trigger ArgMax execution
Inject g_2 during the comparison/update operation of W_t
Attempt to force or preserve the temporary index corresponding to t
Inject g_1 during the loop-control operation
Return the current max_idx as the final predicted label $\hat{y}$
if $\hat{y} = t$ **then**
    return Targeted redirection success, $\hat{y}$
else if the device resets or returns an invalid result **then**
    return Crash, $\hat{y}$
else
    return Targeted redirection failure, $\hat{y}$
end if

exits early, forcing t as the output without any additional faults. However, this condition is difficult for the adversary to exploit in practice because the target index t is not always the local maximum, thus output redirection cannot be consistently achieved.

Attack Point 2: Maximum Comparison/Update. This point corresponds to the instructions that compare the current score with the stored maximum and update max_val and max_idx when the condition is satisfied. A voltage glitch injected at this point may corrupt the comparison result or the conditional branch, causing an erroneous update in the target iteration. However, a single fault at this point is usually insufficient for arbitrary targeted misclassification. Since the ArgMax routine continues to scan later elements, the corrupted max_idx may be overwritten by subsequent correct updates, especially if a later class has a larger score.

Individually, the two attack points provide only limited controllability. To overcome this limitation, we introduce a dual-point voltage fault injection strategy, as summarized in Alg. 2. To redirect the output to a chosen class t, the adversary first uses the ArgMax power trace to locate the iteration window W_t corresponding to t. A glitch g_2 is injected at Attack Point 2 to corrupt the comparison/update behavior and force or preserve max_idx toward the target index t. Another glitch g_1 is injected at Attack Point 1 to terminate the traversal before later iterations can overwrite the corrupted max_idx. In this way, side-channel information provides timing anchors for fault alignment, while the two coordinated glitches jointly enable adversary-chosen output redirection.

D. Shuffled-ArgMax Countermeasures

Based on the identified side-channel and fault-injection vulnerabilities of ArgMax, we propose `Shuffled-ArgMax`, a lightweight countermeasure for protecting the final decision stage, as illustrated in Fig. 5. `Shuffled-ArgMax` consists of three defense mechanisms. Countermeasure 1 randomizes the ArgMax traversal order to disrupt side-channel alignment, while Countermeasures 2 and 3 introduce integrity checks against loop-control faults and maximum-comparison/update faults, respectively.

Countermeasure 1: Fisher–Yates-Based Shuffling. The first vulnerability exploited by our attack is the deterministic traversal order of sequential ArgMax. In a conventional implementation, the output vector is scanned in a fixed order, allowing the adversary to map loop-iteration patterns in the power trace to specific class indices. To break this stable mapping, `Shuffled-ArgMax` randomly permutes the processing order of output indices before each ArgMax execution.

Specifically, we adopt the Fisher–Yates shuffling algorithm, as shown in Alg. 3. Fisher–Yates generates an unbiased random permutation through simple in-place swaps over the index array, making it suitable for resource-constrained embedded devices. After shuffling, the ArgMax routine still visits all K output scores and returns the correct maximum index, but the temporal order of class processing changes across executions. Therefore, the adversary can no longer reliably associate a fixed waveform segment with a specific class index.

This countermeasure directly mitigates attack surface a.1 by removing the deterministic index-to-time mapping. Although repetitive loop patterns and conditional update features may still appear in the power trace, corresponding to attack surfaces a.2 and a.3, their semantic meaning becomes randomized across runs. As a result, the adversary cannot reliably recover the ArgMax decision trajectory or localize the target-class iteration for subsequent fault injection. Thus, Fisher–Yates-based shuffling mainly disrupts trace alignment and decision-timing inference, rather than eliminating all side-channel leakage.

Countermeasure 2: Dual-Counter. To mitigate attack surface b.1, i.e., loop-control faults, we introduce a dual-counter mechanism to verify the integrity of ArgMax traversal. As shown in Fig. 5, the original loop counter i controls the traversal of the shuffled index array, while an auxiliary counter j independently records the number of completed iterations. At the end of ArgMax execution, j is checked against the expected number of classes K. If $j \neq K$, the system treats the execution as potentially faulted and triggers a defense response.

This mechanism is designed to detect premature loop termination. In an unprotected implementation, a voltage glitch may corrupt the loop-control logic and force the routine to exit before all output scores have been processed. With the dual-counter check, even if the main loop counter i or its termination condition is faulted, the auxiliary counter j still

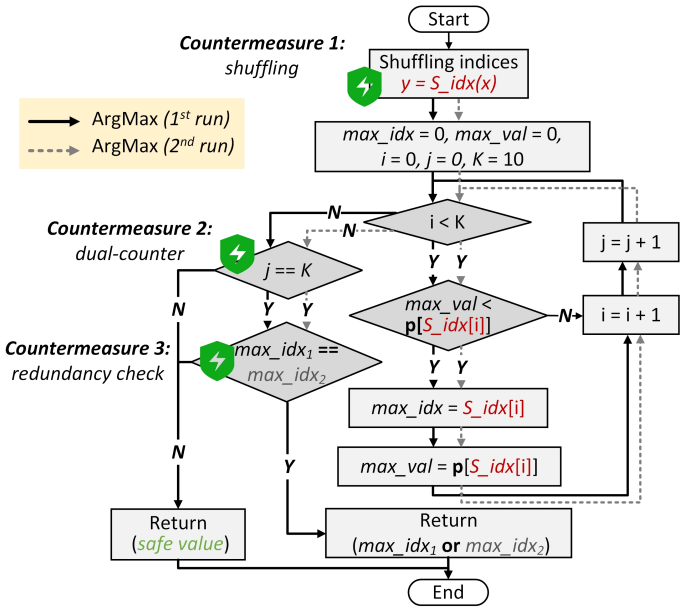


Fig. 5. Overview of `Shuffled-ArgMax`, including traversal randomization, dual-counter loop checking, and redundant decision verification.

reflects the actual number of completed iterations. Therefore, an early exit leaves $j < K$, allowing the system to detect the abnormal traversal.

Bypassing this defense requires the adversary to corrupt not only the loop-control behavior, but also the auxiliary counting or final consistency check. This substantially increases the required fault precision, because the attacker must preserve a valid-looking iteration count while simultaneously inducing an incorrect early termination. Moreover, due to the shuffled traversal order, the timing relationship between the target class, loop-control operation, and integrity check changes across executions, making stable fault injection more difficult. Additional redundant counters or consistency checks can further strengthen this mechanism when a higher security margin is required. In our implementation, one auxiliary counter is sufficient to detect the loop-control faults considered in this work with limited overhead.

Countermeasure 3: Redundancy Check. To mitigate attack surface b.2, i.e., maximum-comparison/update faults, we introduce a redundant decision check for the ArgMax result. The key idea is to execute ArgMax twice on the same output-score vector and compare the two resulting indices, denoted as max_idx_1 and max_idx_2. If the two indices are identical, the result is accepted as the final prediction. Otherwise, the system treats the execution as potentially faulted and triggers a defense response.

Both ArgMax executions are combined with the shuffling mechanism, so the two traversals process the same scores in different randomized orders. This prevents the redundant execution from simply repeating the same temporal leakage and fault window. As a result, a fault that corrupts the comparison or update logic in one traversal is likely to produce an inconsistent result in the other traversal. To bypass this check, the adversary would need to induce consistent

Algorithm 3 `Shuffled-ArgMax` With Fisher–Yates Traversal Randomization

Require: A probability distribution vector $\mathbf{p}$ over K classes
Ensure: Index of the maximum value

Initialize a shuffled index array $\mathbf{S_idx} = [0, 1, \ldots, K-1]$
for $i \leftarrow K-1$ **to** 1 **do**
  Randomly select $j \leftarrow \text{Uniform}(0, i)$
  Swap $\mathbf{S_idx}[i]$ and $\mathbf{S_idx}[j]$ // Fisher–Yates shuffle
end for
$max_idx \leftarrow -1$
$max_val \leftarrow -\infty$
for $t \leftarrow 0$ **to** $K-1$ **do**
  $i \leftarrow \mathbf{S_idx}[t]$
  if $\mathbf{p}[i] > max_val$ **then**
    $max_val \leftarrow \mathbf{p}[i]$
    $max_idx \leftarrow i$
  end if
end for
return max_idx

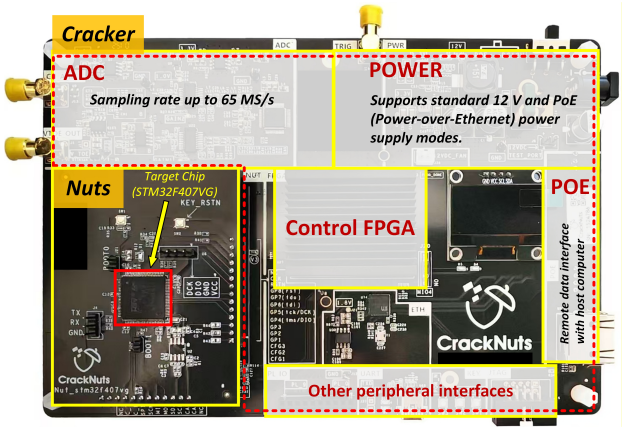


Fig. 6. Experimental platform used for ArgMax power-trace acquisition and voltage-glitch injection.

erroneous outputs across two independently shuffled ArgMax executions, which substantially increases the difficulty of trace alignment and fault timing control under the considered attack model.

V. Experiments and Results

A. Experimental Platform Setup

All experiments are conducted on CrackNuts, an open-source hardware-security evaluation platform supporting both power-trace acquisition and voltage-glitch injection, as shown in Fig. 6. CrackNuts consists of two modular boards: *Cracker* and *Nuts*. The *Nuts* board serves as a swappable target board for STM32-series microcontrollers. In our experiments, it hosts an STM32F407VG MCU running a LeNet-based convolutional neural network (CNN) deployed using the X-CUBE-AI toolchain for MNIST 10-class digit classification. The target MCU operates at 16 MHz.

The *Cracker* board provides experiment control, trace acquisition, and fault-injection capabilities. Its control FPGA coordinates target triggering and attack timing, while the host

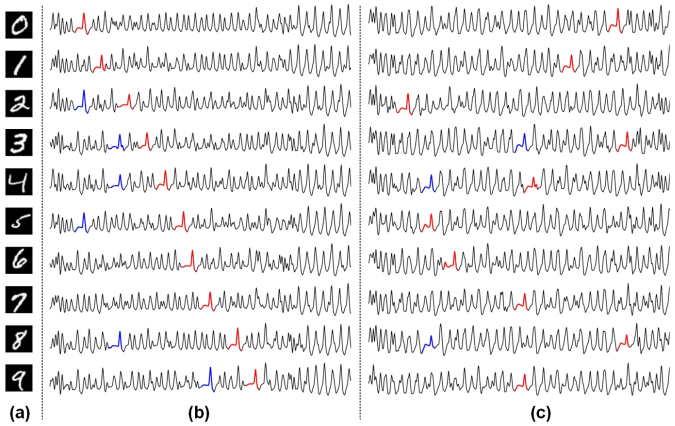


Fig. 7. Representative ArgMax power traces for output recovery: (a) example MNIST inputs, (b) traces without protection showing stable branch-update patterns, and (c) traces with shuffled traversal showing disrupted temporal alignment.

PC configures experiment parameters and collects measurement data. During inference, the on-board ADC samples power traces at 65 MS/s to observe the ArgMax execution. For fault injection, programmable voltage glitches are applied through the target supply path to perturb the ArgMax computation. In our glitch search, the voltage and timing resolutions are set to 0.1 V and 5 ns, respectively. These settings are used throughout the evaluation of the proposed side-channel-guided fault-injection attack.

B. Experimental Results and Analysis

1) Output Recovery Attack and Countermeasure: To validate the proposed output-recovery method, we randomly select one image sample from each digit class in the MNIST dataset, as shown in Fig. 7(a), and collect power traces during ArgMax execution. Fig. 7(b) shows the traces of the unprotected ArgMax implementation, while Fig. 7(c) shows the traces after applying the Fisher–Yates shuffling countermeasure.

As shown in Fig. 7(b), the unprotected ArgMax traces exhibit regular and repetitive structures. The periodic patterns correspond to the loop iterations of the sequential ArgMax routine, while the highlighted jump regions indicate conditional branch-update events. These observations are consistent with the attack-surface analysis in Section IV-B: the deterministic traversal order enables index-to-time mapping (a.1), the repetitive loop structure creates recognizable iteration signatures (a.2), and the conditional maximum update introduces distinguishable timing and power features (a.3). By exploiting these features through SPA, an adversary can reconstruct the ArgMax decision trajectory and recover the final output label of the DNN inference.

In contrast, Fig. 7(c) shows the power traces when only the Fisher–Yates shuffling defense is applied. In this case, the ArgMax iterations are executed in a randomized order. Although repetitive loop patterns may still be visible in the trace, their correspondence to the original class indices is no longer fixed. Consequently, branch-update events no longer reveal a stable decision trajectory in the original index order.

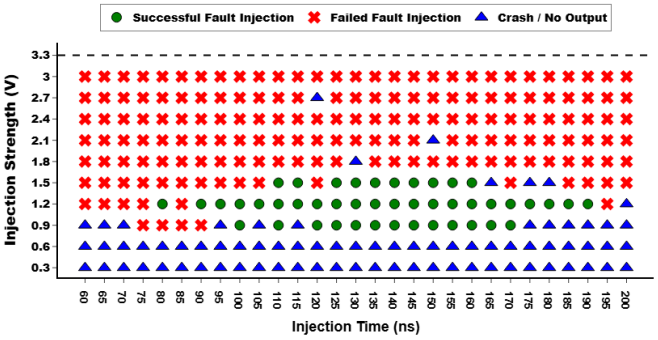


Fig. 8. Glitch-outcome map of the unprotected ArgMax implementation under different injection times and voltages, showing successful redirection, safe execution, and crash regions.

This disrupts the key assumption used by the SPA-based recovery method, namely the deterministic mapping between trace positions and class indices. Therefore, shuffling makes it difficult for the adversary to reliably infer the ArgMax output or localize the target-class iteration for subsequent fault injection.

2) Targeted Misclassification Attack and Countermeasure: We first evaluate the fault sensitivity of the unprotected ArgMax implementation. Fig. 8 shows the glitch-outcome map under different injection timings and glitch voltages. The horizontal axis denotes the injection time relative to the ArgMax execution window, while the vertical axis denotes the applied glitch voltage. The nominal supply voltage of the target MCU is 3.3 V, which is marked as the reference level in the figure. A lower glitch voltage corresponds to a stronger voltage disturbance, as it creates a larger transient supply drop during the injection window.

The glitch-outcome map shows that successful redirection occurs only within a bounded parameter region. For a fixed glitch voltage, increasing the injection duration initially enlarges the effective fault window, with successful outcomes mainly observed between 90 ns and 190 ns. When the duration exceeds this range, the injection either fails to produce the desired effect or destabilizes the execution, leading to crashes. Similarly, for a fixed injection duration, lowering the glitch voltage increases the disturbance strength and can improve the probability of successful redirection. In our experiments, the effective voltage range is mainly between 0.9 V and 1.5 V. When the voltage drops below 0.9 V, the disturbance becomes too strong and frequently causes system crashes. These results indicate that voltage fault injection against ArgMax requires careful parameter tuning: weak glitches fail to affect the decision logic, whereas overly strong glitches destabilize the MCU execution.

We further evaluate the proposed defense under tuned injection parameters, i.e., a glitch voltage of 1.1 V and a duration of 140 ns. For both the protected and unprotected ArgMax implementations, 5000 fault-injection attempts are performed. Four outcomes are recorded: successful targeted redirection, attack failure, defense-triggered protection, and system crash. As shown in Table II, the unprotected implementation produces

TABLE II
OUTCOME DISTRIBUTION OF GUIDED FAULT INJECTION WITH AND WITHOUT SHUFFLED-ARGMAX

Countermeasures	Attempts	Trace Count / Occurrence Probability (%)			
		Success	*Fail*	*Protection*	*Crash*
Unprotected	5000	2846 (56.92%)	8 (0.16%)	N/A	2146 (42.92%)
Protected	5000	0 (0%)	2545 (50.90%)	401 (8.02%)	2054 (41.08%)

TABLE III
TARGET-WISE REDIRECTION SUCCESS RATE ON UNPROTECTED ARGMAX FOR A REPRESENTATIVE INPUT CLASSIFIED AS "6"

Redirect Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Rate (%)	58.7	58.4	58.0	57.3	54.9	59.5	–	53.3	55.1	53.9

2846 successful targeted redirections out of 5000 attempts, corresponding to a success rate of 56.92%. This result confirms that the final ArgMax decision can be externally steered under the local physical-adversary model considered in this work. However, the crash rate is also high, reaching 42.92%, which indicates that the voltage-glitching attack is not fully stealthy and operates within a narrow stability margin.

With `Shuffled-ArgMax` enabled, no successful targeted redirection is observed under the same tested setting. The defense logic is triggered in 401 out of 5000 attempts, corresponding to 8.02%, while the remaining faulty attempts either fail to redirect the output or lead to crashes. This behavior is consistent with the intended role of the proposed countermeasures: they disrupt the stable trace-to-index mapping and increase the difficulty of synchronizing the dual-glitch attack with the target ArgMax iteration. Therefore, the evaluated attack attempts are converted from controlled output redirection into failure, protection-triggered responses, or crashes.

Finally, we assess the target controllability of the combined attack using a representative input image correctly classified as "6". For each target class, 1000 attacks are performed. As reported in Table III, the attack can redirect the output to all non-original target classes in this case study, with broadly comparable success rates across classes. These results demonstrate adversary-chosen redirection capability for the evaluated representative input, rather than merely random misclassification.

3) Runtime Overhead of Shuffled-ArgMax: To assess the performance impact of the proposed `Shuffled-ArgMax` defense, we profile its runtime components, as summarized in Table IV. The shuffling operation introduces the dominant overhead, mainly due to the pseudo-random number generation and in-place index swaps required by the Fisher–Yates permutation. In contrast, the Dual-Counter integrity check and the final Redundancy Check add only minor latency compared with the baseline ArgMax execution.

Although the Redundancy Check executes ArgMax twice using independently shuffled index sequences, the total latency of `Shuffled-ArgMax` is approximately 275 µs. This overhead remains small compared with the end-to-end inference time of the X-CUBE-AI-generated DNN, which is approximately 214 ms. Therefore, `Shuffled-ArgMax` contributes only about 0.13% of the overall inference runtime.

TABLE IV
RUNTIME OVERHEAD OF SHUFFLED-ARGMAX ON A 16-MHZ MCU

Stage	**Time**	**Relative to Baseline**
ArgMax (baseline)	**35.208 μs**	1.00×
Shuffling	98.542 μs	2.80×
Dual-Counter	4.688 μs	0.13×
Redundancy Check	9.167 μs	0.26×
`Shuffled-ArgMax`	**274.688 μs**	7.80×
Overall DNN runtime	**214 ms**	–

These results indicate that the proposed decision-stage hardening mechanisms mainly increase the local ArgMax latency, while introducing only a small end-to-end performance penalty in the present implementation. Combined with the attack results under the tested setting, this suggests that lightweight decision-stage protection can be integrated into resource-constrained edge-AI platforms with low runtime cost.

VI. DISCUSSION

The current evaluation is conducted on a single STM32F407VG-based edge-AI platform and should be viewed as a proof-of-concept on a real embedded IoT target. The identified vulnerability is not tied to a specific MCU peripheral or board-level artifact, but arises from common properties of sequential ArgMax logic, including deterministic traversal, repeated comparison, and data-dependent maximum-index updates. Since similar behaviors also appear in other embedded edge-AI implementations that use sequential score comparison and index selection, the attack surface discussed here may extend to a broader range of hardware platforms.

Although this work uses a CNN on MNIST as the case study, the attack target is the ArgMax-based final decision stage rather than the convolutional layers or the dataset itself. Any model that produces a score vector and applies sequential ArgMax logic may expose a similar decision-stage attack surface. As the number of classes increases, the ArgMax trace becomes longer and the target-iteration search space changes, which may affect both attack calibration and defense effectiveness. Therefore, the 10-class setting should be regarded as an interpretable experimental simplification, while larger class spaces are left for future work.

The observed crash rate indicates that voltage glitching is not fully stealthy and operates within a narrow stability margin. Weak glitches may fail to affect the decision logic, whereas overly strong glitches can destabilize the MCU execution. The main security implication is that side-channel-guided timing information creates a practical window for controlled decision redirection. From a defense perspective,

the key objective is to prevent the adversary from obtaining a valid adversary-chosen output. Under the tested setting, `Shuffled-ArgMax` reduces the targeted redirection success rate from 56.92% to 0%, eliminating successful targeted redirection in our experiments. Therefore, converting controlled redirection into failure, protection-triggered responses, or crashes effectively reduces the adversary's ability to obtain a valid targeted output.

`Shuffled-ArgMax` does not claim to eliminate all side-channel leakage. Its goal is to break the fixed correspondence between class index and trace position, thereby disrupting target-iteration localization and fault injection. Its effectiveness depends on the unpredictability of the shuffled traversal order. An adaptive adversary may attempt to collect more traces, infer the shuffle order, or attack the random-number generation and integrity-check logic. Future work will investigate such adaptive attacks and combine `Shuffled-ArgMax` with stronger randomness generation, fault sensors, or repeated inference verification.

VII. Conclusion and Future Work

This work presented a systematic security analysis of the ArgMax unit, the final decision stage in DNN inference. We showed that sequential ArgMax logic can expose exploitable power-leakage patterns due to deterministic traversal, repeated comparison, and data-dependent maximum updates. Based on this observation, we developed a side-channel-guided voltage fault injection attack that recovers decision-related timing information and enables targeted misclassification to an adversary-chosen class. To mitigate this threat, we proposed `Shuffled-ArgMax`, a lightweight defense that combines traversal randomization, loop-integrity checking, and redundant decision verification. We implemented and evaluated the proposed attack and defense on the CrackNuts platform using a CNN deployed on an STM32F407 MCU. The results show that ArgMax decision logic in embedded edge-AI systems can be externally manipulated under a local physical-adversary model, while `Shuffled-ArgMax` disrupts the evaluated attack path with only a small end-to-end runtime overhead. These findings indicate that ArgMax should not be treated as benign post-processing, especially in physically exposed edge-AI IoT devices where final predictions may directly trigger safety-relevant actions.

For future work, we plan to extend the analysis to other post-processing operations commonly used in DNN deployments, such as Top-k selection, which may expose similar side-channel or fault-injection attack surfaces. We will also investigate ArgMax implementations on other hardware platforms, including FPGA and ASIC accelerators, and evaluate their security under side-channel attacks, fault injection attacks, and combined physical attacks. Another promising direction is to study more advanced adversaries, such as those combining electromagnetic observation with voltage fault injection or targeting the randomization and integrity-checking logic of the defense. Overall, these directions further suggest that decision-stage security deserves closer attention in embedded and edge-AI IoT inference systems.

References

[1] M. Méndez Real and R. Salvador, "Physical side-channel attacks on embedded neural networks: A survey," *Applied Sciences*, vol. 11, no. 15, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/15/6790

[2] K. Yoshida, T. Kubota, S. Okura, M. Shiozaki, and T. Fujino, "Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[3] L. Wu, L. Wu, and X. Zhang, "Catch the star: Weight recovery attack using side-channel star map against dnn accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 10, pp. 3697–3709, 2025.

[4] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532.

[5] A. Kurian, A. Dubey, F. Yaman, and A. Aysu, "Tpuxtract: An exhaustive hyperparameter extraction framework," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2025, no. 1, p. 78–103, Dec. 2024. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11923

[6] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, "Stealing neural network structure through remote fpga side-channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[7] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. New York, NY, USA: Association for Computing Machinery, 2018, p. 393–406.

[8] Z. Wang, Y. Wu, Y. Park, S. Yoo, X. Wang, J. K. Eshraghian, and W. D. Lu, "Powergan: A machine learning approach for power side-channel attack on compute-in-memory accelerators," *Advanced Intelligent Systems*, vol. 5, no. 12, p. 2300313, 2023.

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

[10] J. Ordoñez and C. Yang, "Derailed: Arbitrarily controlling dnn outputs with targeted fault injection attacks," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[11] Y. Fukuda, K. Yoshida, and T. Fujino, "Fault injection attacks utilizing waveform pattern matching against neural networks processing on microcontroller," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 105, no. 3, pp. 300–310, 2022.

[12] A. A. Malik, F. Aydin, and A. Aysu, "Precision strike: Targeted misclassification of accelerated CNNs with a single clock glitch," Cryptology ePrint Archive, Paper 2025/1728, 2025. [Online]. Available: https://eprint.iacr.org/2025/1728

[13] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power side-channel attacks on bnn accelerators in remote fpgas," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 357–370, 2021.

[14] K. Yoshida, T. Kubota, S. Okura, M. Shiozaki, and T. Fujino, "Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

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

[16] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 295–300.

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

[18] B. Goswami, R. Chetry, C. Moorthii J, and M. Suri, "Investigation of em fault injection on emerging lightweight neural network hardware," in *Applied Cryptography and Network Security Workshops*, M. Manulis, Ed. Cham: Springer Nature Switzerland, 2026, pp. 113–123.

[19] M. Dumont, K. Hector, P.-A. Moellic, J.-M. Dutertre, and S. Pontié, "Evaluation of parameter-based attacks against embedded neural networks with laser injection," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2023, pp. 259–272.

[20] S. Lee, S. Kim, S. Hong, and J. Ha, "Clock glitch fault attacks on deep neural networks and their countermeasures," *Sensors*, vol. 25, no. 9, p. 2793, 2025.

[21] J.-S. Coron, "A new dpa countermeasure based on permutation tables," in *International Conference on Security and Cryptography for Networks*. Springer, 2008, pp. 278–292.

[22] M. Brosch, M. Probst, and G. Sigl, "Counteract side-channel analysis of neural networks by shuffling," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1305–1310.

[23] L. Puškáč, M. Benovič, J. Breier, and X. Hou, "Make shuffling great again: A side-channel-resistant fisher–yates algorithm for protecting neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 7, pp. 1959–1971, 2025.

[24] S. Shukla, M. Alam, S. Bhattacharya, P. Mitra, and D. Mukhopadhyay, ""whispering mlaas": Exploiting timing channels to compromise user privacy in deep neural networks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 2, p. 587–613, Mar. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/10295

[25] B. Coqueret, M. Carbone, O. Sentieys, and G. Zaid, "When side-channel attacks break the black-box property of embedded artificial intelligence," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 127–138. [Online]. Available: https://doi.org/10.1145/3605764.3623903

[26] Z. Liu, S. van Hoek, P. Horváth, D. Lauret, X. Xu, and L. Batina, "Real-world edge neural network implementations leak private interactions through physical side channel," 2025. [Online]. Available: https://arxiv.org/abs/2501.14512

[27] W. G. Shin, J. Lee, S. Jung, and H. Kim, "Armor: First-order masking of activation and argmax gadgets for side-channel resistant neural networks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2026, no. 2, p. 106–131, Apr. 2026. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/12882