Robust Stealthy High-Impact Malicious Hardware Attacks on Deep Neural Networks

Maath Frman , Kholood J. Moulood , Mustafa Noori , Ekram H. Hasan , Oqbah Salim Atiyah , Qutaiba Alasad *

Posted Date: 5 June 2026

doi: 10.20944/preprints202606.0498.v1

Keywords: deep learning; hardware Trojans; machine learning; neural networks accelerators; hardware security; zeroing and sign-flipping attacks

 doi:10.20944/preprints202606.0498.v1

Article

Robust Stealthy High-Impact Malicious Hardware Attacks on Deep Neural Networks

Maath Frman [1], Kholood J. Moulood [2], Mustafa Noori [3], Ekram H. Hasan [4], Oqbah Salim Atiyah [5] and Qutaiba Alasad [6,*]

[1] Department of Computer Science, Tikrit University, Salah al-Din, Iraq
[2] Department of Math, Tikrit University, Salah al-Din, Iraq
[3] Department of Artificial Intelligence, Tikrit University, Salah al-Din, Iraq
[4] Department of First Grades Teacher, University of Anbar, Ramadi, Iraq
[5] Department of Computer Science, Tikrit University, Salah al-Din, Iraq
[6] Departments of Cybersecurity and Petroleum Systems Control Engineering, Tikrit University, Salah al-Din, Iraq
* Correspondence: qutaibaeng@tu.edu.iq

Abstract

The rapid advancement of modern Deep Neural Networks (DNNs) has played a crucial role in aiding humans in many real-world applications, yet their hardware accelerators have been proven to be vulnerable to malicious attacks. One particularly severe and serious attack involves inserting a hardware Trojan (HT) into DNN accelerator hardware in order to enable attackers to stealthily manipulate model predictions during the supply chain. In this paper, we present a *possible* stealthy HT architecture that is difficult to be detected and has a significant impact on the performance of DNN models. To successfully achieve this goal, we introduce the Sensitivity-Based Weight Selection (SBWS) algorithm, a novel technique that adapts machine learning (ML) sensitivity analysis to identify and modify only a small number of weights that have the highest impact on DNN performance, compared to previous work. We evaluate the proposed attack on five DNN model tests (two distinct DNN models and four different datasets) using two designed payload types: weight zeroing and sign-flipping, and record the results based on various security metrics. The experimental results show average accuracy reductions of 26.7% for the zeroing attack and 48.1% for the sign-flipping attack, yielding an overall average of 37.4%, calculated over five independent runs per dataset with standard deviation $< 2\,\%$. The sign-flipping technique consistently outperforms the zeroing one because it preserves the magnitudes of the attacked weights while inverting their signs, thereby disrupting the learned decision boundaries more severely and amplifying error propagation in subsequent layers. These results significantly exceed previous random-weight perturbation attacks (typically 12–20% drops) and other targeted HT approaches, while incurring lower computational and hardware resource overheads. This work provides a more effective and scalable method for assessing the vulnerability of DNN accelerators under real supply-chain threat models.

Keywords: deep learning; hardware Trojans; machine learning; neural networks accelerators; hardware security; zeroing and sign-flipping attacks

1. Introduction

In contemporary research and applications, neural networks (NNs) have been considered as a prevalent choice for executing different tasks, including object detection, image classification, and speech recognition [1–4]. DNN models have become valuable forms of intellectual property (IP) for many businesses. They were first used only for basic tasks, such as image classification in on-site facilities (local systems). However, nowadays they have been applied in many more important applications, including the traffic monitoring [5], autonomous driving [6], medical diagnostics [7], and weather prediction and forecasting [8]. Currently, it is expected that the utilization of Artificial Intelligence (AI)

 doi:10.20944/preprints202606.0498.v1

will be continued to experience substantial global market growth in the coming years [9]. Despite their significant accomplishments and noticeable benefits, the increasing dependence on the hardware DNN accelerators to achieve real-time and energy-efficient inference in the limited resource environments, e.g., the medical diagnostics, edge devices, and autonomous vehicles, has introduced severe security vulnerabilities from both of the general hardware trust perspectives [10] and the DNN-specific hardware vulnerabilities [11,12]. In fact, DNN accelerators, often implemented utilizing third-party IP cores and fabricated using global supply chains, have been shown to be particularly susceptible to hardware Trojans (HTs). The HTs are malicious hardware circuit modifications inserted during the integrated circuit (IC) design or fabrication process, which can stealthily decrease model accuracy or leak sensitive information [13,14]. Many defensive techniques against various HTs have been presented in order to protect digital IC systems and networks from serious malicious intrusions [7,8].

DNNs outperform human beings in terms of providing very high accuracy and performance in many applications; however, they require a large number of parameters, and this significantly elevates the memory usage, power consumption, and system complexity. Unfortunately, the high required power consumption and cost in server farms prevent the design from being fabricated on a real chip. Therefore, there is a need to shift from server-based to the local infrastructure, leveraging mobile devices, tablets, drones, and autonomous systems in order to achieve real-time processing and improve both data privacy at lower energy dissipation [15–17].

Deep Learning Accelerators (DLAs) are promising techniques that are integrated into electronic devices, including resource-limited Internet of Things (IoT) devices. These incorporate many components into a single IC [18], including the controllers, processors, memory, network-on-chip, converters, and input/output (I/O) units [19]. The main goal of the DLAs is to reduce the power dissipation and cost compared to software platforms. However, due to the complexity of the design, validation, and manufacturing processes, DLAs frequently depend on global supply chains, and unfortunately, this exposes them to many hardware security issues [20,21]. Growing hardware-level threats to DNN accelerators include various HTs, fault injections, and memory-based manipulations, which can stealthily compromise the model integrity without being detected [22,23]. Not that software-level attacks, e.g., adversarial examples, have been well studied, but hardware-oriented threats have not been fully explored [11,24]. Despite their significant accomplishments and noticeable benefits, the NN reliability has become a major challenge. Adversarial attacks can threaten the reliability of NNs by exploiting input vulnerabilities in order to mislead them and produce inaccurate outputs. Moreover, Third-party IPs may contain malicious components, such as HTs and their variations, e.g., backdoors, fault induction, memory trojaning, and interconnection tampering [10–12]. Therefore, ensuring DLA security is essential because hardware vulnerabilities can significantly affect the reliability of the system.

Several HT attacks targeting the DNN accelerators have been introduced in recent years. The early works focused mainly on inserting Trojans into FPGA-based CNN accelerators by modifying the activation parameters or utilizing the rare input patterns as triggers [25]. The memory Trojan attacks [26] and noise injection into ReLU activations [27] were later introduced in order to decrease the model accuracy or leak sensitive information. Other approaches have targeted the off-chip memory or leveraged the statistical analysis to detect or insert serious hardware Trojans. Even though these techniques have demonstrated the feasibility of HT attacks on the DNN hardware, they have generally suffered from one or more of the following limitations: (i) relatively moderate accuracy or performance reduction (typically between 15 and 35%), (ii) requiring high hardware overhead due to the complex designed trigger logic or the large required numbers of the modified elements, (iii) less stealthy triggers with higher activation probability, and (iv) non-targeted or random weight perturbations, which render them fail to correctly exploit the structural sensitivity of the DNN weights. These shortcomings decrease both the practical effect and the stealthiness of the previously presented attacks in real supply-chain threat models.

In this work, two critical observations have motivated us, as follows: (1) existing HT attacks on DNN accelerators are not achieved to highly reduce the performance of the model (typically between

 doi:10.20944/preprints202606.0498.v1

20 and 30% reduction) or require a large number of modifications, which leads to render the malicious circuit be detected due to the required high hardware overhead or by using side-channel signatures; (2) most of previous methods depend on the random, non-targeted, or broadly distributed perturbations, which in turn fail to exploit the structural sensitivity of DNN weights, especially in early layers where small changes can be propagated and amplified through the network. These limitations reduce both the practical impact and the stealthiness of the attacks, limiting their ability to be real threat models against robust modern accelerators. To address these gaps, the following main contributions have been presented in this work:

- We have proposed the SBWS, a novel and scalable algorithm that adapts the gradient-based sensitivity analysis in order to select the smallest possible number of the most affected weights to achieve up to 2× higher reduction efficiency per modification than the random or non-targeted approaches.
- We design and evaluate a lightweight possible stealthy HT architecture that leverages the SBWS-selected weights and activates only under rare net-signal conditions using the simple logic gates (NAND trigger + configurable payload for zeroing or sign-flipping).
- The experimental results have been conducted on five DNN model tests (two distinct DNN models and four different datasets), and the results demonstrate that this proposed HT can reduce the DNN accuracy by an average of 26.7% for weight zeroing and 48.1% for sign-flipping (overall 37.4%). This outperforms random perturbations (12–20% drops) and other previously presented HTs, with ultra-low overhead (0.3–0.4%) and extremely rare trigger probability (10^{-5}). Such strong HT ensures the real need for a strong and robust detection technique in supply-chain hardware implementations.

The proposed technique has several unique features, as follows: (i) a novel SBWS algorithm is implemented to identify only a smaller possible number of the highly influential weights (typically 5–7); (ii) each Trojan can accomplish significant accuracy reduction (26.7% with weight zeroing and 48.1% with sign-flipping) while incurring ultra-low hardware overhead (0.3–0.4%); (iii) the trigger circuit is extremely rare (10^{-5} probability), rendering each proposed Trojan very stealthy; and (iv) the sign-flipping consistently outperforms zeroing attack because it can carefully preserve the weight magnitude while significantly disrupting the learned decision boundaries. These features make the proposed Trojans effective and practical under real supply-chain threat models.

This paper has been organized as follows: Section 2 provides background on DNNs and related HT attacks. Section 3 presents the proposed possible HT architecture in detail, including the SBWS algorithm, threat model, and circuit-level insertion. Section 4 shows the experimental setup, results, and discussion, including the comparison with related work. Finally, the conclusion and future work are given in Section 5.

2. Background and Related Work Analysis

2.1. Analysis of Deep Neural Networks (DNNs)

DNNs mainly comprise enormous interconnected neuron layers that can process their inputs. Each neuron applies a function to its inputs and then forwards the outputs to incoming (subsequent) [1]. The Input layers feed the hidden layers, in which neurons aggregate weighted inputs and then apply the used activation functions. The common activations include the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ (0 to 1 range), the $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (-1 to 1), and the $\text{SoftMax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ for the probabilities [28,29]. During the training phase, the weights of DNNs are repeatedly updated by utilizing backpropagation and gradient descent functions in order to minimize the loss and obtain the most possible and accurate output [30,31].

 doi:10.20944/preprints202606.0498.v1

2.2. Analysis of Hardware Trojans Insertion into DNN

Due to proven vulnerabilities in DNNs, many hardware attacks, including reverse engineering, HTs, and side-channel attacks (SCA), have been presented. In this paper, we have mainly focused on a malicious insertion via the HTs because they represent a proactive and embedded threat that permanently alters the functionality of the DNN hardware, and this, in turn, makes them stealthier and more severe in supply-chain scenarios compared to others. The detection evasion is further refined by polymorphic triggers, e.g., varying based on model hyperparameters, and a minimal footprint.

HTs, also called malicious insertion, are unauthorized hardware manipulations that compromise the security of a hardware circuit by leaking sensitive information [32,33]. The purposes of HTs are to either decrease the performance or trigger the denial-of-service. They consist of the trigger, which is activated under rare conditions, and the payload, which represents the malicious effect or action [34]. Triggers use rare signals in the original hardware design, and the payloads modify the circuit's function. HTs can be in different places, including the transistor, gate, bus, or IP levels [35]. Detecting them is very difficult using traditional verification tools [32,36,37]. Unlike reverse engineering-based attacks [38], which primarily expose hardware designs for copying or replication without direct runtime impact, or Side Channel Attacks (SCAs) [39], which exploit observable emissions and can often be mitigated through shielding or monitoring the design, HTs enable undetectable small added hardware circuit that can effectively degrade the DNN accuracy during the normal operation. This paper is mainly focused on designing a targeted HT insertion method using the SBWS algorithm, which significantly reduces the performance of the DNN accelerator without being detected. Recent works ensure that HTs are considered more severe than others due to their ability to evade different traditional verification methods and directly compromise critical hardware systems.

2.3. Analysis of Literature Review

HTs are malicious IC modifications that are inserted by attackers before manufacturing the IC in order to leak information or reveal DNN accelerators' functionality [13].

Current studies are grouped into trigger-based attacks, where rare conditions have been employed to activate payloads. For example, HTs can be embedded in DNNs via using supply-chain adversaries [14]; or HTs can be inserted into Field-Programmable Gate Array (FPGA)-based CNNs using Multiply–Accumulate-Based Matrix Architecture (MAT) architecture [25]. These trigger techniques aim to remain stealthy by activating them only under specific rare inputs or conditions. However, they are often limited by practical implementation due to the difficulty in designing reliable controllable rare triggers, which can be potentially detected through using the side-channel monitoring of activation patterns, and due to increasing hardware overhead when the complex trigger logic is required [40]. Also, the memory trojaning technique has been shown to negatively impact DNN's performance without being detected [23].

Payload-based attacks directly impact attacked hardware designs. For instance, they can reduce the accuracy via DNN services [41], inject a malicious without incorporating input disturbances [42]; employ statistical analysis [43]; attack an off-chip memory [26]; insert noise into ReLU activation functions [27]; target Static Random-Access Memory (SRAM) in caches [13]. Such payload approaches concentrate on persistent or direct reduction and manipulation without depending on explicit input triggers. Even though they are effective in reducing the performance or leaking sensitive information [40], many of these methods incur noticeable performance overhead, provide limited fine-grained control over the degree of degradation, or are subject to statistical anomaly detection and runtime monitoring techniques.

Moreover, other studies have further clarified the effectiveness of these threats on DNN models, as follows: provide a comprehensive evaluation on HT attacks and defenses on DNN architectures [44], explore resilient CNN accelerators against different HT attacks [45], and propose Siamese NNs to effectively detect HTs using side-channels [46]. These contributions have highly advanced the understanding of HT vulnerabilities in many DNN accelerators and proposed different countermeasures.

 doi:10.20944/preprints202606.0498.v1

Nevertheless, attack and defense techniques focus on broad or generic HT insertion and detection mechanisms, which leads to suboptimal reduction efficiency per the given modification, limited scalability on different accelerator architectures, and insufficient focus on the targeted model with smaller possible modification-based attacks that can be used to highly elevate the accuracy loss and preserve the stealthiness at low overhead.

Overall, although previous research has shown the feasibility and impact of HTs on DNN accelerators through utilizing both trigger and payload mechanisms, there are still significant limitations in achieving high performance reductions with smaller possible changes and ultra-low overhead, including wider applications on accelerator designs. In this paper, we propose SBWS, a systematic algorithm for selecting the most sensitive weights, with a stealthily designed HT. The proposal outperforms other previous methods by achieving higher degradation efficiency with minimal possible modifications and overhead, which in turn addresses current gaps in the targeted and scalable attacks presented in recent related works.

In order to clearly clarify research gaps and place this work, we compare related HT attacks on DNN accelerators with the proposal in different terms, including trigger mechanism, payload type, reported accuracy drops, modifications, overhead, stealth (trigger rarity), and attacker knowledge required. Table 1 briefly summarizes this comparison. The previous techniques have typically achieved moderate reduction in the design's performance (20–35%) with higher required overhead (0.5–1.5%) and less rare triggers (10^{-3}–10^{-4}). The terms "Moderate" and "Variable" in the table correspond to 20–35% accuracy reduction and 0.5–1.5% area/power overhead, respectively. However, the SBWS-based technique attacks only the most sensitive weights to accomplish significantly higher dropped performance with smaller possible changes (only 5–7 weights modified), ultra-low overhead (0.3–0.4%), and extremely rare trigger activations (10^{-5} probability). This design renders the possible HT architecture efficient, scalable, and stealthy.

Table 1. Comparison between this proposal and other related works.

Reference	Trigger Mechanism	Payload Type	Main Limitations/Gaps	Attacker Knowledge Required
	# Mod./Overhead (%)	Acc. drop (%)	Stealth (Trigger Prob.)	
[37]	Rare input patterns (FPGA MAT) Multiple gates/logic	Activation param. modification 15–35	High overhead; less targeted 10^{-4}	Partial
[42]	Input-triggered 32 bits/variable	Memory trojaning 20–35	Moderate degradation; memory-specific 10^{-3}–10^{-4}	Partial
[43]	Rare conditions 15 elements	Noise in ReLU 25	Noticeable overhead; anomaly detectable 10^{-3}	Partial
[41]	Statistical rare signals Low–moderate	Input interception Variable	Limited control; statistical detection risk Moderate	Partial
[13]	Rare cache access Low	SRAM cache targeting Moderate	Cache-specific; runtime monitoring vulnerable Moderate	Partial
[44]	Survey (various) Variable	Various 20–38 (typical)	Broad/generic; suboptimal per-mod efficiency Variable	Variable
[45]	Survey/resilient designs Variable	Various 28	Focus on defense; limited attack targeting 10^{-3}	Variable
This proposal	Rare low-activity nets 5–7 weights	Weight zeroing/sign-flipping 26.7–48.1	Targeted, minimal mods, high efficiency, low overhead 10^{-5}	Partial (netlist)

3. The Proposed Attack

3.1. Modifying the Most Impact Neurons

Modifying weights in the early hidden layers of a given DNN propagates its effects throughout the entire network, which significantly impacts the overall accuracy. Figure 1 shows the effectiveness of

 doi:10.20944/preprints202606.0498.v1

modifying weights to incoming layers. This cascading effect usually occurs because the output of each neuron serves as input to the next incoming layers, which can highly amplify the initial perturbations. Mathematically, the output of a neuron can be represented by $o = f(\sum_i w_i x_i + b)$, where the f is the activation function, w_i are weights, x_i are primary inputs, and b is a bias of the network. Changing a single value, e.g., w_i, in an early layer influences all of the downstream computations and can potentially lead to a substantial drop in the accuracy when the modification is strategically selected. This principle underpins the main basis of the proposed attack since it can produce a larger possible disruption in the performance by making smaller possible changes.

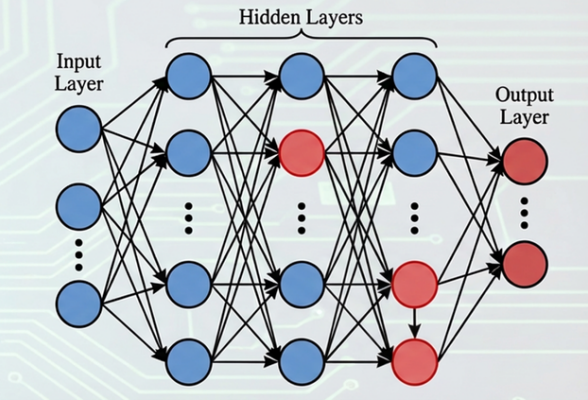

Figure 1. Illustration of the cascading impact when modifying a single neuron in a hidden layer of a DNN, in which perturbations are propagated through the network and impact subsequent layers and final outputs.

3.2. Threat Model

As mentioned before, the main purpose of designing this proposed HT attack is to degrade the accuracy of DNNs by targeting and manipulating the most affected weights in the model during the design or fabrication phase. This has been modeled as a white-box attack, in which adversaries have complete knowledge about the architecture and structure of the DNN, e.g., layer structure, neuron connections, and its hyperparameters, e.g., weight and bias values. Such knowledge allows the precise SBWS algorithm targeting and rare-trigger selection. However, they do not have access to or knowledge about the training and testing datasets. Also, note that the supply-chain compromise assumes that the legitimate owner has no visibility into the malicious modifications inserted by untrusted third-party vendors or insiders. Therefore, the owner will perform standard functional verification on the unmodified specification, while the Trojan is still dormant and undetectable during the testing.

Capabilities of the adversary include the ability to adjust the values of the weights and biases during the design, manufacturing, or implementation phases of the DNN accelerator hardware. During these phases, the attacker can carefully and stealthily insert an HT that should be activated only under rare and predefined conditions, e.g., specific input signals or patterns, thereby flipping the weight signs or resetting them to zero without being triggered during the standard verification processes. In fact, this introduced threat model reflects the real-world scenarios and applications, such as:

- Insider threats: in which a malicious contractor or employee can access the design files and modify the hardware netlist.

- Supply-chain compromises: The third-party fabrication facilities or vendors carefully embed the hardware Trojan during the global IC production without the owner's knowledge.

- Outsourcing vulnerabilities: For example, when DNN IP cores are sourced from untrusted vendors (e.g., via platforms like GitHub or commercial IP marketplaces), attackers can pre-insert Trojans before integration.

- Post-deployment updates: Adversaries could exploit firmware updates in edge devices to inject Trojans via over-the-air mechanisms.

 doi:10.20944/preprints202606.0498.v1

The main assumptions include: (1) The attacker cannot modify runtime data flows post-deployment to avoid detection by integrity checks; (2) Standard testing covers 99% of common inputs, but rare triggers (e.g., specific pixel patterns in images) are untested; (3) The DNN is deployed on hardware like FPGAs or Application-Specific Integrated Circuits (ASICs) without real-time weight monitoring. Note that accessing the data is restricted by the privacy regulations, e.g., the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA), or by the security policies, and this, in turn, limits the attacker's ability to exploit the model's architecture rather than the dataset itself. This setup emphasizes real practical vulnerabilities in DNN implementations and highlights the need for more robust hardware security techniques.

3.3. Algorithm for Identifying Most Sensitive Weights

In order to systematically identify the most affected weights ("sensitive weights"), we propose the Sensitivity-Based Weight Selection (SBWS) algorithm that is demonstrated in Algorithm 1. Our novel presented algorithm has been implemented based on the sensitivity analysis approaches from the machine learning, such as the gradient-based saliency maps in adversarial attacks and the importance scoring in the pruning techniques, and they are uniquely adapted to be used for hardware Trojan insertion into the DNN accelerators. By iteratively grouping weights and evaluating the reduced accuracy of the model when they are modified, the SBWS algorithm efficiently isolates the smallest set of weights that can cause the most significant performance drops in the performance. This targeted approach reduces the Trojan's hardware footprint, e.g., fewer hardware logic gates are needed without being detected. This algorithm outperforms the non-targeted and the random selection technique in previous related works by achieving up to 2X higher reduction efficiency per the modified weight, as will be shown in the experiments Section 4.

Algorithm 1 Sensitivity-Based Weight Selection (SBWS).

Require: DNN model M, validation dataset D, modification type mod ("zero" or "flip"), initial group size $initial_G$, minimum group size min_G, degradation threshold θ
Ensure: Set of sensitive weights S

$acc_{\text{original}} \leftarrow \text{Acc}(M, D)$ ▷ Compute baseline accuracy
$S \leftarrow \emptyset$
for each layer L in M **do**
  $weights_L \leftarrow$ extract weights from L
  $groups \leftarrow$ divide $weights_L$ into groups $\{G_1, \ldots, G_n\}$ where $|G_i| = initial_G$
  $candidates \leftarrow \emptyset$
  for each G_i in $groups$ **do**
    $Temp_M \leftarrow$ copy of M
    modify G_i in $Temp_M$ according to mod ▷ set to zero or negate the sign
    $acc_{\text{drop}} \leftarrow acc_{\text{original}} - \text{Acc}(Temp_M, D)$
    if $acc_{\text{drop}} > \theta$ **then**
      add G_i to $candidates$
    end if
  end for
  while $|candidates| > 0$ **and** $\min\{|G_i| : G_i \in candidates\} > min_G$ **do**
    $new\_candidates \leftarrow \emptyset$
    for each G_i in $candidates$ **do**
      $subgroups \leftarrow$ subdivide G_i into $\{Gs_1, \ldots, Gs_m\}$ where $|Gs_j| = |G_i|/2$
      for each Gs_j in $subgroups$ **do**
        $Temp_M \leftarrow$ copy of M
        modify Gs_j in $Temp_M$ according to mod
        $acc_{\text{drop}} \leftarrow acc_{\text{original}} - \text{Acc}(Temp_M, D)$
        if $acc_{\text{drop}} > \theta$ **then**
          add Gs_j to $new\_candidates$
        end if
      end for
    end for
    $candidates \leftarrow new\_candidates$
  end while
  $S \leftarrow S \cup$ (union of all weights in $candidates$)
end for
return S

 doi:10.20944/preprints202606.0498.v1

The algorithm starts by processing the DNN layer-wise, and taking the advantages of the network's hierarchical structure, in which small changes in the early layers strongly impact many incoming downstream computations in the next layers. It employs a degradation threshold θ which is experimentally tuned via using the grid search on the validation sets, e.g., 0.5% in our experiments, in order to prune the low-impact groups and ensure maintaining the design scalable. For models with millions of hyperparameters (N), we have used the divide-and-conquer technique to further reduce the computational complexity from $O(N)$ (exhaustive per-weight testing) to $O(\log N \times G)$, where the G represents the initial number of the groups.

Moreover, the SBWS algorithm supports two modification techniques: zeroing (setting the values of the weights to 0, mimicking a hardware reset) or flipping the sign of the weights (multiplying by -1 to invert the sign of a given weight and retain the magnitude). The experimental results indicate that flipping the sign of the weights often yields a substantial reduction, as it disrupts the learned optimization directions without removing the connections entirely.

It is worth mentioning that one possible limitation of the group-based dichotomy search is that some affected weight combinations on different initial groups may not be fully explored. However, because the algorithm keeps any subgroup whose impact exceeds the threshold θ and because the perturbations in early layers propagate globally, the selected weights will still have a high impact in the search process. The ablation studies show only a small difference from the exhaustive search on small models and significantly decrease the search time.

Following the weight selection, we further analyze the DNN hardware implementation to find the storage positions for the sensitive set S, and this has been done utilizing the Synopsys Verdi tools for the netlist inspection and simulation. Before we start with inserting the two designed Trojans into the DNN netlist, we consider a simple implementation of these Trojans, shown on the left side of Figure 2. Both attacks utilize a combination logic gates of triggers and payloads in order to maliciously manipulate the state of the circuit. Normally, the Trojans are inactive, and the design works correctly, e.g., the set and reset signals of the D flip-flops (DFFs) are '0'. The trigger network is activated when the selected rare combination of signals (n_0 to n_N) drives the NAND gate, which (in combination with the set and reset control signal) will force the XNOR gate output to '1'. For zeroing Attack, once triggered, the payload will drive the reset signal of the DFF to be flipped in order to switch the stored value to '0'. For the sign flipping attack, when triggered, the payload will drive the set signal of the DFF to be '1' and, in this case, the Most Significant Bit (MSB) of an architecture's weight will be flipped to a negative sign (set to '1'). The designed hardware Trojan can then be carefully inserted during the untrusted stages of the DNN chip fabrication process, shown on the right side of Figure 2. The rare net—signals that are extremely low switching during the normal operations are served as triggers. These rare nets have been fed to an XNOR gate. This will make the Trojan be activated only when a condition on the rare signals is satisfied. More specifically, the sign flipping attack will flip the most significant bit (MSB) of the selected DNN weight registers, which will flip the sign of the corresponding weights. This manipulation will significantly decrease the performance of the DNN mode or produce wrong classification without showing any immediate functional failures. For the zeroing attack, the targeted DFFs will be reset by the Trojan, which will corrupt the stored weight values in the DNN accelerator.

The proposed Trojan architecture has been designed using a small number of gates, which require very small area, power, and delay overheads, and therefore cannot be easily detected through the design-time verification. Furthermore, because the Trojan is obscure and can only be activated when the rarely signal patterns are set, it is highly resistant to the conventional functional testing, the random input stimulation, and the side-channel analysis techniques. As a result, the Trojan cannot be easily detected during the post-fabrication validation and will cause a severe impact after the DNN chip is fabricated.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 June 2026 doi:10.20944/preprints202606.0498.v1

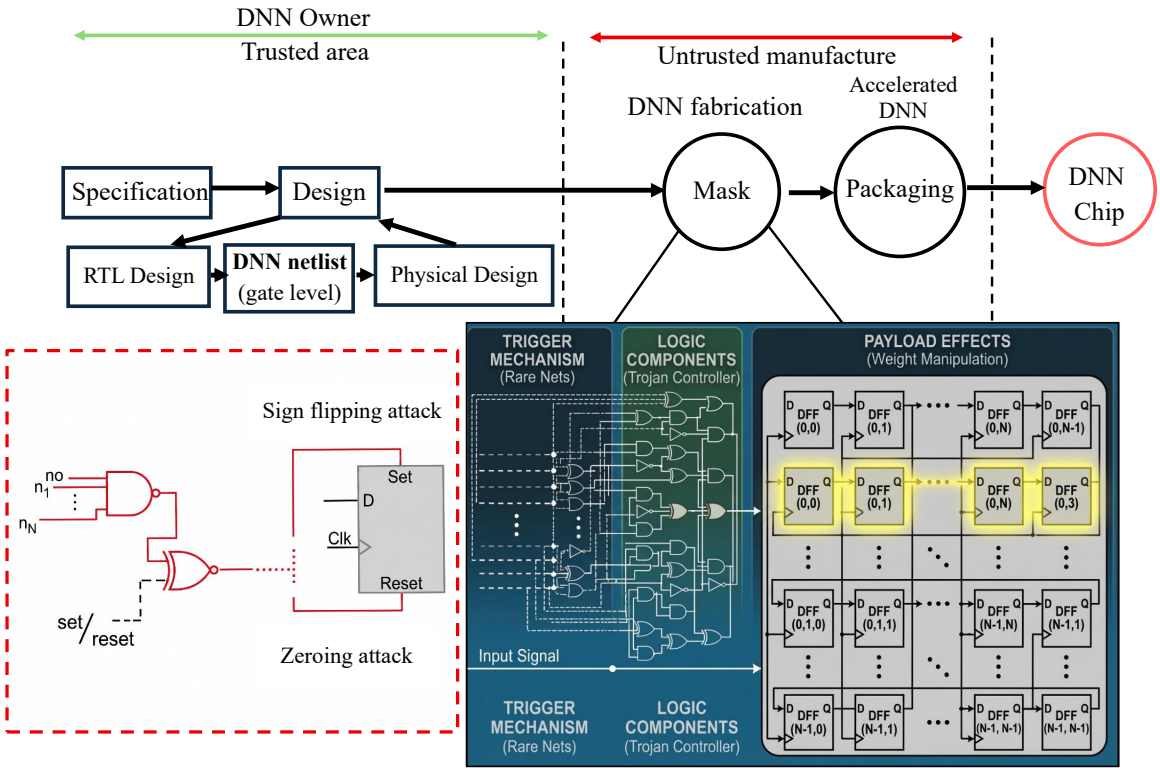

Figure 2. Framework of the proposed possible hardware Trojan architecture, illustrating the trigger mechanism (rare nets), logic components (XNOR gate and DFFs), Trojan controller, and payload effects on selected weights.

3.4. Insertion of the Designed Hardware Trojan

The hardware Trojan can be inserted during the hardware design or fabrication phase of the DNN accelerator, specifically targeting the on-chip memory or register files where weights are stored, e.g., SRAM arrays or flip-flop-based buffers in the processing elements.The low-activity (rare-signal) nets are identified post-synthesis in the weight-storage datapath utilizing the netlist analysis. These nets are deliberately chosen from non-critical paths (with timing slack greater than 20% of the clock period) in order to ensure that the added Trojan logic gates do not introduce detectable timing or power side effects. Because the weights selected by the SBWS algorithm have been mapped to specific register files, their corresponding read ports naturally provide the rare signal nets required for triggering. The design choice will directly couple the trigger technique to the SBWS output without any conflict, since the trigger selection is operated on the read ports of the already-selected weight registers. As a result, the Trojan will achieve both high attack impact and strong stealthiness against conventional detection approaches. The insertion of the designed HT process involves the following two steps:

1. **Netlist Modification:** Post-synthesis, add the Trojan logic (NAND and XNOR gates for trigger and payload) between the weight read port and the multiply-accumulate (MAC) units in the accelerator's computation pipeline. For zeroing, connect to the clear input of DFFs storing weights. For sign-flipping, use a XOR on the MSB of floating-point weights (IEEE 754 format).
2. **Placement in Supply Chain:** The Trojan can be embedded by a malicious foundry during RTL-to-GDSII conversion or by an insider in the IP integration stage, ensuring it bypasses pre-silicon verification by activating only under post-deployment rare conditions (e.g., specific input data patterns not in test benches). For evasion, we select triggers from non-critical paths to avoid power/timing anomalies detectable by side-channel analysis (e.g., $< 0.1\%$ power increase).

This targeted insertion in early-layer weight storage can increase propagation effects while maintaining stealth, as it avoids altering control signals that can trigger the side-channel detectors. The detection evasion is further enhanced by the polymorphic triggers, e.g., varying based on model

 doi:10.20944/preprints202606.0498.v1

hyperparameters, and a smaller footprint, making it resistant to tools like the Trust-Hub benchmarks or the ML-based anomaly detectors.

The trigger circuits have been placed on the non-critical paths (slack $>$ 20% of the clock period, verified with the PrimeTime static timing analysis) and consist of the low-activity nets ($<$ 1% toggle rate during 10^6 random input simulations). These nets are identified in the post-synthesis utilizing the netlist analysis in the weight-storage datapath and have been directly coupled to the SBWS-selected weight registers, which will, in turn, ensure a higher possible impact with lower observability.

3.5. Illustrative Example

In order to clearly demonstrate the core idea of the attack, we provide a simple example employing a small neural network that implements an XNOR gate, shown in Figure 3 and Table 2. In this example, a simple trained neural network is considered to implement an XNOR gate, which explains the main idea of this work. After applying 10,000 training iterations with a sigmoid activation, the given model achieves 100% accuracy. By flipping the sign of specific weights, e.g., the second upper weight in layer zero, the accuracy has been reduced by 25, while by applying the same setup to the next lower weights, the accuracy has been dropped to 50%. In larger and more complex models that have millions of hyperparameters, such targeted modifications become exceedingly difficult to be detected during the verification process since they blend into normal parameter variability (hide within the normal parameter noise), which highlights the stealth and effectiveness of the proposed approach.

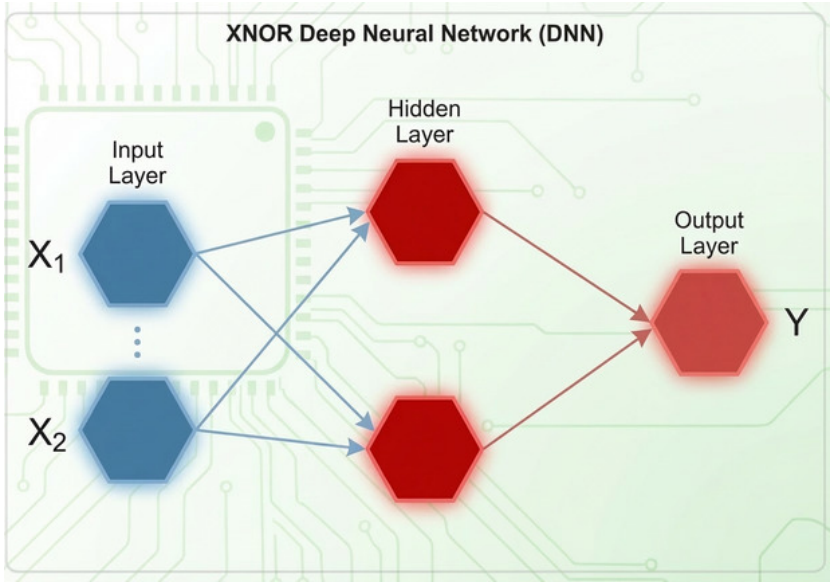

Figure 3. Example demonstrating the manipulation impact on weights of a single node in a simple XNOR neural network, showing the pre- and post-attack accuracy changes.

Moreover, the weights are fundamentally used to determine and control the model's output, and the strategic tampering enabled by SBWS can efficiently degrade the reliability of the model without overt signs of compromise. Note that, in this example, we leverage 64-bit floating-point (IEEE 754 double) weights in order to clarify the presentation. In real hardware NN implementations, especially on embedded and edge devices, much lower precision is typically used, such as 8-bit integers (INT8), 16-bit floating-point (FP16 or bfloat16), or even lower-bit fixed-point and quantized representations. The attack principle demonstrated here is still applicable to the aforementioned lower-precision formats, even though the exact sensitivity of individual weights may vary with the chosen quantization scheme. We have discussed this further in the in Section 4.

 doi:10.20944/preprints202606.0498.v1

Table 2. Weights of the Motivation Example (XNOR Neural Network).

Layer Number	Weights Index	Weights Values	Binary Representation (IEEE 754 Double)
Layer zero	–	-5.879879499206042	1100000000000101100110101001111110010001011000011101001010011101
Layer zero	–	6.077031263144357	0100000000001100100101100001011000010011001100011010100001110001
Layer zero	–	6.827562389096342	0100000000001101010110010101111110010001011000011101001010011101
Layer zero	–	-6.58329960234238	1100000000001101000010110111111110010001011000011101001010011101
Layer one	–	9.471175496792553	0100000000010010110100001001111110010001011000011101001010011101
Layer one	–	9.33944757890564	0100000000010010110010010001111110010001011000011101001010011101

4. Results and Discussion

4.1. Experimental Setup

All of the experiments have been conducted utilizing the Jupyter Notebook version 7.0 on a high-performance server that is equipped with an NVIDIA RTX 4080 SUPER GPU (16 GB GDDR6X VRAM), 32 GB system RAM, and the Intel Core i7 processor operating at 2.66 GHz base frequency (with turbo boost up to 4.6 GHz). This setup provides sufficient computational resources for both training and evaluating the DNN models in order to effectively handle the datasets and attack simulations.

The employed datasets include different real-world DNN application scenarios, ranging from the high-dimensional image classification to the low-dimensional tabular medical data, which allows us to comprehensively assess the proposed attack's generalization. The Date Fruit dataset (multiclass image classification) has been obtained from Kaggle (Murat KOKLU et al., 2021) and contains approximately 900 high-quality images for 7 date fruit varieties (Barhee, Deglet Nour, Sukkary, Rotab Mozafati, Ruthana, Safawi, and Sagai). which is captured in the controlled computer vision systems. The Images have been preprocessed with the standard resizing, e.g., 224×224 for CNNs, and augmented with the random flips and rotations during the training phase. The HT attack has been tested on the color image features with moderate complexity. The UCI Heart Disease dataset is a tabular binary classification obtained from the UCI ML Repository. It is based on the original Cleveland database and consists of 303 instances with 13 features, including age, sex, cholesterol, chest pain type, etc.). It can be used to predict the presence and the absence of the heart disease. The dataset contains both the numerical and categorical features, which are normalized to the [0,1] range, and it has been selected due to its low dimensionality and high real-world impact in the medical diagnostics, where even small accuracy drops can lead to serious consequences. The Fashion MNIST, which is a standard multiclass grayscale image classification dataset released by Zalando Research. It includes 60,000 training and 10,000 test 28×28 grayscale images, each with a resolution of 28 X 28 pixels, to cover 10 clothing classes: the T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. No heavy preprocessing has been applied beyond the normalization to the [0,1] range, and it has been used to evaluate the attack performance on the distributed lower-resolution image representations. The fashion CNN variant is built on the Fashion MNIST dataset using a convolutional architecture, e.g., Conv2D layers + pooling + dense layers, to represent the modern accelerator-targeted CNNs. Finally, the Histopathologic Cancer Detection, which is a binary medical image classification dataset obtained from the Kaggle competition, consists of about 220,000 labeled histopathology image patches extracted from the larger whole-slide images, used to classify the metastatic cancer presence or absence in the small tissue patches and regions. The preprocessing with resizing and augmentation has been applied

 doi:10.20944/preprints202606.0498.v1

to this dataset, and we have chosen it due to its high-stakes medical domain and its large-scale, high-resolution image data that allows us to test the scalability of the proposed attack. We also use different models for the evaluation purposes, as follows: Multilayer Perceptron (MLP) and CNN variants, trained to obtain the baseline accuracies leveraging the standard optimizers, e.g., the Adam optimizer, and the loss functions, e.g., the cross-entropy. The accuracy has been calculated as: $\text{Acc} = \frac{\text{TP+TN}}{\text{Total}}$, while the absolute accuracy drop is computed as: $\Delta\text{Acc} = \text{Acc}_{\text{orig}} - \text{Acc}_{\text{attacked}}$.

4.2. Results

Table 3 shows the results for the zeroing attack, where the selected weights have been reset to zero, in which the hardware reset via D flip-flops (DFFs) is simulated. The average reported absolute accuracy drop is 26.7%, which has been calculated as the mean of individual drops: 26.1%, 33.0%, 5.7%, 36.7%, 32.1%. This demonstrates that the SBWS algorithm is very efficient in identifying the most affected weights, with only a few attacks needed, even for complex datasets, including the Heart Disease, where the results show that only changing 4 neurons drops the accuracy by 33%. It is worth noting that the variation in the attacked neurons reflects the dataset complexity, in which the high-dimensional image data, e.g., Fashion MNIST, requires more targets due to the distributed representations, while the tabular data, such as the Heart Disease, allows the concentrated impact. The presented results highlight the vulnerability of the DNN accelerator in practical applications, such as medical diagnostics (the heart disease and histopathologic cancer detection), in which even very small modifications can lead to substantial misclassifications. This can potentially endanger patient outcomes or the reliability of the system. Moreover, the lower reduction in the Fashion MNIST dataset, 5.7%, suggests that the grayscale image datasets with the distributed features might require more targeted neurons to effectively achieve high performance reduction, and this in turn underscores the real need for the adaptive attack techniques based on the data dimensionality.

Table 3. Impact of setting sensitive weights to 0 (using SBWS).

Model on Dataset Type	Train/Test Split	Orig. Acc	# Neurons Attacked	Acc After Attack
CNN on Date Fruit	900 / 300	91.1%	7	65%
MLP on Heart Disease	242 / 61	89%	4	56%
MLP on Fashion-MNIST	60k / 10k	88.7%	12	83%
CNN on Fashion-MNIST	60k / 10k	92.7%	6	56%
CNN on Histopath. Cancer	220k patches	91.9%	5	59.8%

We also report the impact of the sign-flip attack, as shown in Table 4, where inverting the weights (w $\rightarrow$ -w) is achieved via flipping the most significant bit (MSB). The average absolute drop is 48.1%, where the individual drops are: 37.1%, 42.0%, 31.7%, 79.9%, 49.6%. Based on the obtained results, it has been pointed out that the sign-flipping achieves higher impact than the zeroing attack because it preserves the weight magnitude while reversing contributions, which mathematically disrupts the decision boundaries more severely, e.g., inverting the gradients in the backpropagation paths. For instance, in the Fashion CNN, flipping only 2 weights causes a catastrophic 79.9% drop in the accuracy, and this highlights the SBWS's ability to target the most critical and important neurons in the convolutional layers. The overall average drop on both attacks is 37.4%, underscoring the effectiveness of the proposed attack. Also, the difference between zeroing and sign-flipping underscores the importance of keeping the magnitude in the attacks. The zeroing completely eliminates the contributions, which, at the same time, allows the network redundancy to partially compensate for the loss. Whereas the sign-flipping keeps the magnitude, but reverses the direction, which exploits the learned model optimizations and causes high values in the downstream layers. Such results indicate broader implications for the hardware security in the smart systems, such as the autonomous vehicles and defense applications, where the sign-flipping attack can induce stealthy failures without being detected. These emphasize the real need for strong defensive techniques.

 doi:10.20944/preprints202606.0498.v1

Table 4. Impact of flipping signs.

Model on Dataset Type	Train/Test Split Dataset	Orig. Acc	# Neurons Attacked	Acc After Attack
CNN on Date Fruit	900 / 300	91.1%	3	54%
MLP on Heart Disease	242 / 61	89%	2	47%
MLP on Fashion-MNIST	60k / 10k	88.7%	12	57%
CNN on Fashion-MNIST	60k / 10k	92.7%	2	12.8%
CNN on Histopath. Cancer	220k patches	91.9%	6	42.27%

Furthermore, we have implemented and applied the Sensitivity-Based Weight Selection (SBWS) algorithm with different hyperparameters, as follows: initial group size $initial_G = 100$ (to balance between the granularity and the computation), minimum group size $min_G = 1$ (for the single-weight precision), and the reduction threshold $\theta = 0.5\%$ (optimized via leveraging the grid search on the validation subset in order to obtain highly impact with the smaller possible runtime). A 20% validation split from each dataset has been used to compute the accuracy during the SBWS algorithm and the post-attack evaluations, and this, in turn, ensures that the results will be unbiased during the assessment. We also evaluate the robustness of the SBWS algorithm by conducting the ablation studies on different θ values (0.1%, 0.5%, 1.0%) across 5 different runs for each value. The lower θ (0.1%) identifies 15% more weights, but the runtime is doubled (average is 2.1x), and this happens due to broader candidate retention. A higher θ (1.0%) reduces the targets by 20%, but unfortunately, it lowers the average degradation by 5–7% since fewer sensitive weights have been captured. The 0.5% threshold optimizes and balances the trade-off as confirmed by the Pareto analysis of impact versus computational cost. Note that all these results are averaged over 5 runs per dataset, with a standard deviation less than 2%. We also provide a comparison between the impact of setting the sensitive weights to zero and flipping the sign of the weight value. Table 5 shows this comparison in detail and reveals that the sign-flipping outperforms the zero attacking by exploiting the learned optimizations.

Table 5. A comparison between the impact of setting the sensitive weights to 0 and flipping the sign of the weight value.

Aspect	Table 3 (Zeroing)	Table 4 (Sign-Flipping)
Impact of Attacks	Moderate negative impact on accuracy (avg 26.7% drop).	Higher impact, with steeper drops (avg 48.1%, e.g., 79.9% in Fashion CNN).
Dataset Variability	Consistent drops (5.7–36.7%), less sensitive to structure.	More variable (31.7–79.9%), amplifying dataset-specific sensitivities.
Vulnerability	Lower overall vulnerability; partial mitigation via weight regularization.	Higher vulnerability due to inversion effects; demands advanced defenses like sign-invariant training.
Model Performance	Less disruptive in dense datasets (e.g., minimal 5.7% in Fashion MNIST).	Significant decreases, especially in CNNs, emphasizing attack potency.
Defense Mechanisms	Suggest robust monitoring of zeroed paths.	Imply issues in protection; calls for gradient masking or anomaly detection.
Generalizability	Applicable to various models with consistent but moderate effects.	Broad impact on architectures, underscoring comprehensive security needs.

The trade-off between average accuracy drop and runtime across varying degradation thresholds θ is calculated and taken into consideration. Figure 4 illustrates the Pareto front for this trade-off. The data points correspond directly to the ablation results: $\theta = 0.1\%$ achieves a high 40.2% drop but at the cost of 120 seconds runtime; $\theta = 0.5\%$ achieves a strong 37.4% drop with a more efficient 65 seconds; and $\theta = 1.0\%$ reduces the drop to 32.1% but runs quickest at 45 seconds. The curve confirms $\theta = 0.5\%$ and $initial_G = 100$ as the optimal balance, increasing impact with smaller computational overhead, as determined via multi-objective optimization. We extend the implementation to initial group size ($initial_G$: 50, 100, 200), finding $initial_G = 100$ optimal (37.4% drop, 65s runtime). Smaller groups

 doi:10.20944/preprints202606.0498.v1

increase precision but runtime (1.5x for 50), while larger groups reduce impact (32% for 200). Table 6 summarizes this Pareto front, confirming $\theta = 0.5\%$, $initial_G = 100$ as the sweet spot via multi-objective optimization. We further implement the SBWS algorithm at different thresholds θ and compare it with the exhaustive per-weight search technique. As explained in Table 7, the SBWS with $\theta = 0.5\%$ accomplishes 37.4% an average accuracy drop, which is close to the 39.8% obtained by exhaustive search, while reducing the runtime by about 12× faster. As a result, elevating θ will further decrease the runtime, but will also decrease the attack effectiveness. This hyperparameter analysis reveals the real practical trade-offs that the attackers encounter in the limited resource environments, such as the embedded systems, where the lower runtime (higher θ) is preferred by the attackers on a very high performance reduction. Moreover, the low standard deviation (<2%) on different runs indicates that the SBWS is robust and reliable on different DNN architectures. Such reliability also points to the need for defenses against such strong attacks. Moreover, in order to assess the effectiveness of the SBWS algorithm, we directly compare it with the random weight selection utilizing the same number of modified weights. As shown in Table 8, attacking the selected weights by the SBWS algorithm will cause significantly higher accuracy degradation than the random selection on all models. In fact, this demonstrates that the sensitivity-guided selection is very critical to achieving a higher possible impact with a smaller possible number of modifications.

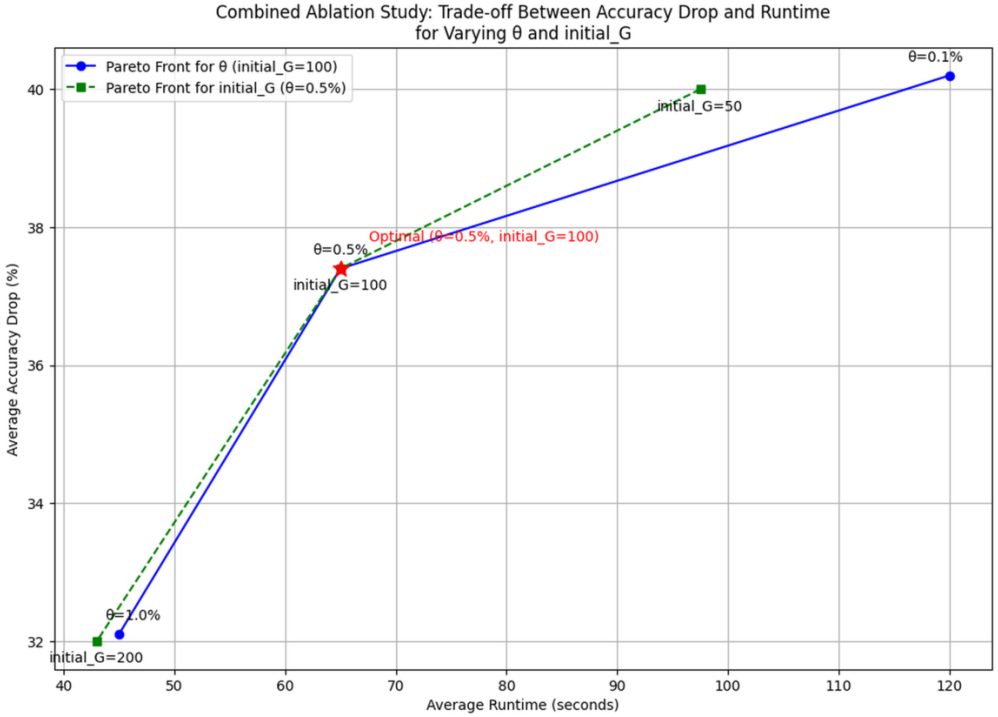

Figure 4. The ablation study plot showing the trade-off between the dropped accuracy and the runtime for varying θ values in the DNNs, balanced with theta = 0.5% and initial_G = 100 to minimize the runtime while achieving high accuracy reduction.

Table 6. Ablation on Initial Group Size ($initial_G$).

$initial_G$	Avg. Acc. Drop (%)	Avg. # Weights	Avg. Runtime (s)
50	38.5	7.0	98
100	37.4	6.2	65
200	32.0	5.1	50

 doi:10.20944/preprints202606.0498.v1

Table 7. Ablation Study on SBWS θ Ablation vs. Exhaustive Per-Weight on small models.

Method	θ	Avg. Acc. Drop (%)	Avg. # Selected Weights	Avg. Runtime (s)
Exh. search	–	39.8	6.5	780
SBWS	0.1%	40.2	8.4	120
SBWS	0.5%	37.4	6.2	65
SBWS	1.0%	32.1	4.7	45

Table 8. Comparing the dropped accuracy between the Random Weight Selection (RWS) and the SBWS technique.

Model on Dataset	# Weights	RWS Acc. Drop (%)	SBWS Acc. Drop (%)
CNN on Date Fruit	7	12.4	26.1
MLP on Heart Disease	4	9.8	33.0
MLP on Fashion-MNIST	12	4.2	5.7
CNN on Fashion-MNIST	6	8.6	36.7
CNN on Histopath. Cancer	5	11.3	32.1
Average	–	**9.3**	**26.7**

In fact, the proposed SBWS enables higher impact on the performance of the model with only a few modifications compared to random weight attacks, where the baseline average drops 12% based on the tests with very small Trojan overhead. It is worthy to mention that the design has been synthesized using the Synopsys Design Compiler on TSMC 45nm library, and results show that the HT incurs very low area overhead (0.4%) and power dissipation (0.3%), which is much better than the ones in the related literature on the DNN HTs, which are typically $< 1\%$ for stealth attacks. To further validate this work, we use the statistical analysis via the paired t-tests, 10 runs per model, n=50 samples, and the p-value < 0.01, and the results confirm a significant reduction in the accuracy. Compared to prior works that included the recent resilient designs [45] and the general noise injection, with 20–30% drops [47], the proposed Trojan is much more stealthier due to carefully selecting the rare triggers and the implemented SBWS-targeted selection, which also evades Siamese NN detection [46] by minimizing side-channel signatures. Note that the low overhead further improves the stealthiness since the SBWS can be integrated into the limited-resource IoT devices without triggering the design verification alarms. The statistical validation ($p < 0.01$) shows that the observed performance reduction is not random, and this implies that the proposed SBWS can be extended to bypass strong defenses, e.g., the runtime monitoring, if it has been combined with the polymorphic triggers.

Even though the proposed attack is significant and powerful, it is important to mention its limitations, where the white-box assumption is used. This may not hold in the fully obfuscated systems, and also gives a lack of evaluation on certain transformers, such as the BERT. Furthermore, fixed-point quantization may mitigate the impact by reducing bit-flip sensitivity, and large-scale models, e.g., GPT-like models, may require scaled SBWS to be feasible. Ethically, this research highlights vulnerabilities to inform defenses, not enable attacks; all experiments were simulated without real hardware deployment to avoid harm. Besides these limitations, this technique requires the white-box access, which makes it less applicable in the black-box attack settings, in which the attackers do not have knowledge about the model architecture. One can carefully address this issue by augmenting the approximation methods, such as the query-based the sensitivity estimation. Also, the impact of this work has not been evaluated on the transformer models since the attention techniques have different sensitivity patterns.

4.3. Performance Hardware Overhead

The added Trojan logic utilizes only a small number of logic gates, e.g., 5-10 gates per targeted weight, resulting in $< 0.5\%$ area overhead and $< 0.3\%$ power overhead, verified via the post-synthesis evaluation using the Synopsys Design Compiler and the PrimeTime on the 45nm Taiwan Semiconductor Manufacturing Company Standard Cell (TSMC) library. Compared to baseline DNN accelerators,

 doi:10.20944/preprints202606.0498.v1

such as the Eyeriss-style accelerator designs, this overhead is negligible, while maintaining the original clock frequency to be at 1 GHz.

4.4. Security Analysis of the Proposed Trojans

It is worth analyzing and showing why the proposed Trojan attacks are difficult to be detected. Generally, there are three main ways to detect an inserted design [39]:

- **Logic testing**– supply the possible input patterns to activate and detect the Trojan. Some of the best tools to run the logic testing are the Automatic Test Pattern Generation (ATPG), controllability and observability analysis, and Heuristic search, e.g., MERO, AdaTest. Unfortunately, since the DNN is very large and complex (with millions of neurons and weights), these techniques cannot detect such a strong Trojan. Similarly, when employing the brute force tools.
- **Hardware performance overhead analysis**– using Synopsis tools to measure how much the performance overhead of the design, e.g., the area, has been increased. However, since the proposed Trojan is very small (it has only a few logic gates compared to the DNN, which has millions of gates), the increased overhead is negligible, and many testing tools will not detect any difference.
- **Identification of the Trojan layout**- during the fabrication process, the location gates of the Trojan will be intermixed with the original DNN circuitry since the logic synthesis and physical design (Place & Route) tools will automatically determine the gate placement during the process of creating the final GDSII layout. Also, the logic synthesis tools apply optimization techniques that will change the types and structures of the logic gates to satisfy the design constraints, such as the power dissipation, area, and timing. Due to the extremely large number of logic gates in modern DNN hardware architectures, recognizing and isolating a sophisticated hardware Trojan within the layout will be highly challenging.

In short, the proposed Trojans are very hard to be detected as the DNN model size is very large, the relative overhead of the Trojan is significantly decreased, fewer gates for the Trojan versus several millions of weights, while the rarity of the trigger remains 10^{-5} because SBWS focuses on a fixed small number of weights (5–12).

4.5. Precision Considerations in Real Hardware Implementations

The illustrative example in Section 3.5 utilizes 64-bit floating-point weights for readability purposes. However, modern DNN accelerators that are implemented on edge devices and embedded systems rarely leverage the full-precision floating-point arithmetic. Instead, they usually employ the low-bit-width representations, including 8-bit integers (INT8), fixed-point quantization, or 16-bit floating-point formats (FP16, bfloat16), in order to highly decrease the memory footprint, power consumption, and latency. Since the core attack technique (identifying the sensitive weights via the SBWS algorithm and applying the sign-flipping or zeroing attack) remains valid on different numeric formats, the effectiveness of the bit-flip-style or sign-flip attacks can vary depending on the quantization approach used and bit-width. For instance, in the highly quantized networks, flipping the sign of a weight may have a more (or less) pronounced effect due to the reduced dynamic range and the presence of the scaling factors. The proposed attack can be assessed under real quantization settings, e.g., post-training INT8 quantization and quantization-aware training, on the NVIDIA Jetson and Google Coral Edge TPU hardware platforms.

4.6. Comparing Analysis with Previous Works

To further highlight this approach, we compare it against the recent HT attacks on DNNs. Table 9 shows a comparison between this proposal and other works leveraging different metrics, including the accuracy reduction, number of modifications, hardware overhead, and stealth, which is used to measure the trigger rarity probability, where the lower values indicate the higher stealthier.

 doi:10.20944/preprints202606.0498.v1

Table 9. Comparison between this proposal and other HT attacks on DNNs, where the results indicate that this method offers better efficiency (higher drop per modification) with lower overhead, compared with other techniques. Note that all previous and this techniques are white-box attacks.

Technique	Avg. Acc. Drop (%)	# Mod.	OH (%)	Stealth (Trigger Prob.)
Clements et al. (2019) [27]	25	15	1.2	10^{-3}
Sun et al. (2025) [45]	28	8	0.6	10^{-3}
Zhao et al. (2019) [26] (Memory T.)	30	32	0.5	10^{-3}
Ye et al. (2018) [25] (FPGA CNN HT)	< 30	8	1.5	10^{-4}
Our SBWS (Zeroing)	26.7	6.8	0.4	10^{-5}
Our SBWS (Sign-Flipping)	48.1	5	0.3	10^{-5}

Note that these drops are model-specific and verified through 10 runs (p less than 0.01 via using t-tests); random perturbations yield about 12-20% drops in accuracy based on the provided tests. However, this method, supported by the SBWS algorithm, achieves higher drops with fewer modifications and lower overhead. Also, the stealth has been improved by selecting rarer triggers, e.g., 1 in 100,000 inputs, and this outperforms noise-injection methods [27], which often require more than 1% overhead and can be detected by utilizing statistical tests. This refinement comes from the proposed SBWS's gradient-based sensitivity analysis, which allows more precise targeting than broader approaches in previous works, resulting in up to 2x better reduction in the efficiency per each modification. Consequently, these results put the proposed Trojan to be considered as a baseline for the future HT designs and expose weaknesses in existing defensive techniques.

5. Conclusions and Future Work

In this work, a hard-to-detect hardware Trojan inserted into five DNN model tests (two distinct DNN models and four different datasets) has been designed in order to attack the performance of DNNs. The experimental results show that the proposed attack reduces the accuracy of models by about ∼37% on average via using the novel SBWS algorithm that is incorporated to select the most sensitive weights. The proposed algorithm provides a proficient and explainable technique to find and attack most affected weights in a network, and also introduces a novel way to implement serious HT attacks. It is worth mentioning that this work emphasizes the real and urgent need to protect and secure supply chains, runtime integrity checks, e.g., hash-based verification, and standardize the AI safety measures to mitigate such serious threats that can be used to attack sensitive and essential hardware applications and systems.

The possible future work will include proposing a detection technique, e.g., runtime monitoring or adversarial training, and will also take into account black-box attack scenarios and modern DL architectures, such as transformers. Involving black-box attacks can be achieved via utilizing approximation techniques and quantifying the detailed hardware overhead across different nodes, e.g., 28nm, and then testing it on more advanced architectures. Additional directions include integrating with federated learning for distributed attacks and developing quantum-resistant defenses against emerging threats.

Author Contributions: Conceptualization, M.F.; methodology, Q.A. and M.F.; software, M.F. and K.J.M.; validation, M.N. and O. S. A.; formal analysis, Q.A. and K.J.M.; investigation, M.N. and M.F.; data curation, E.H.H. and O. S. A.; writing—original draft preparation, Q.A. and M.F.; writing—review and editing, Q. A., M.F., and E.H.H.; supervision, Q. A. All authors have read and approved the final manuscript.

Funding: This research has received no external funding.

Data Availability Statement: No new data have been created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: Not applicable

Conflicts of Interest: The authors declare no conflicts of interest.

 doi:10.20944/preprints202606.0498.v1

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* **2017**, *60*, 84–90. https://doi.org/10.1145/3065386.
2. Mikolov, T.; Deoras, A.; Povey, D.; Burget, L.; Cernocky, J. Strategies for Training Large Scale Neural Network Language Models. In Proceedings of the 2011 IEEE Workshop on Automatic Speech Recognition & Understanding. IEEE, 2011, pp. 196–201. https://doi.org/10.1109/ASRU.2011.6163930.
3. Hinton, G.; Deng, L.; Yu, D.; Dahl, G.E.; rahman Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.N.; et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* **2012**, *29*, 82–97. https://doi.org/10.1109/MSP.2012.2205597.
4. Abolghasemi, P.; Mazaheri, A.; Shah, M.; Boloni, L. Pay attention!-robustifying a deep visuomotor policy through task-focused visual attention. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, 2019, pp. 4254–4262. https://doi.org/10.1109/CVPR.2019.00439.
5. Srivastava, S.; Narayan, S.; Mittal, S. A survey of deep learning techniques for vehicle detection from UAV images. *Journal of Systems Architecture* **2021**, *117*, 102152. https://doi.org/10.1016/j.sysarc.2021.102152.
6. Luo, M.; Myers, A.C.; Suh, G.E. Stealthy tracking of autonomous vehicles with cache side channels. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 2020, pp. 859–876.
7. Fatehi, N.; et al. Towards Adversarial Attacks for Clinical Document Classification. *Electronics* **2023**, *12*, 129. https://doi.org/10.3390/electronics12010129.
8. Alahmed, S.; et al. Mitigation of Black-Box Attacks on Intrusion Detection Systems-Based ML. *Computers* **2022**, *11*, 115. https://doi.org/10.3390/computers11070115.
9. Artificial Intelligence Market Growing at a CAGR of 36.6% and Expected to Reach $190.61 Billion by 2025. PR Newswire, 2019.
10. Hu, W.; Chang, C.H.; Sengupta, A.; Bhunia, S.; Kastner, R.; Li, H. An Overview of Hardware Security and Trust: Threats, Countermeasures, and Design Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2020**, *40*, 1010–1038.
11. Xu, Q.; Arafin, M.T.; Qu, G. Security of Neural Networks from Hardware Perspective: A Survey and Beyond. In Proceedings of the Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC), 2021, pp. 449–454.
12. Zhou, T.; Zhang, Y.; Duan, S.; Luo, Y.; Xu, X. Deep Neural Network Security from a Hardware Perspective. In Proceedings of the Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH). IEEE, 2021, pp. 1–6.
13. Wang, X.; Hoque, T.; Basak, A.; Karam, R.; Hu, W.; Qin, M.; Mu, D.; Bhunia, S. Hardware Trojan Attack in Embedded Memory. *J. Emerg. Technol. Comput. Syst.* **2021**, *17*. https://doi.org/10.1145/3422353.
14. Mittal, S.; Gupta, H.; Srivastava, S. A survey on hardware security of DNN models and accelerators. *J. Syst. Archit.* **2021**, *117*. https://doi.org/10.1016/j.sysarc.2021.102163.
15. Farabet, C.; Martini, B.; Akselrod, P.; Talay, S.; LeCun, Y.; Culurciello, E. Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of the Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 257–260. https://doi.org/10.1109/ISCAS.2010.5537908.
16. Dally, B. Hardware for Deep Learning. In Proceedings of the 2023 IEEE Hot Chips 35 Symposium (HCS), 2023, pp. 1–58. https://doi.org/10.1109/HCS59251.2023.10254716.
17. Nabavinejad, S.M.; Baharloo, M.; Chen, K.C.; Palesi, M.; Kogel, T.; Ebrahimi, M. An overview of efficient interconnection networks for deep neural network accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **2020**, *10*, 268–282. https://doi.org/10.1109/JETCAS.2020.3007705.
18. Tamir, A.; et al. Multi-Tier 3D IC Physical Design with Analytical Quadratic Partitioning Algorithm Using 2D P&R Tool. *Electronics* **2021**, *10*, 1930. https://doi.org/10.3390/electronics10161930.
19. Wu, Y.N.; Emer, J.S.; Sze, V. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019, pp. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942149.
20. Sumathi, G.; Srivani, L.; Murthy, D.T.; Madhusoodanan, K.; Murty, S.A.V.S. A review on HT attacks in PLD and ASIC designs with potential defence solutions. *IETE Technical Review* **2018**, *35*, 64–77. https://doi.org/10.1080/02564602.2016.1268059.

 doi:10.20944/preprints202606.0498.v1

21. Zhang, J.; Qu, G. Recent Attacks and Defenses on FPGA-based Systems. *ACM Trans. Reconfigurable Technol. Syst.* **2019**, *12*. https://doi.org/10.1145/3340557.
22. Breier, J.; Hou, X.; Jap, D.; Ma, L.; Bhasin, S.; Liu, Y. Practical Fault Attack on Deep Neural Networks. In Proceedings of the Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2018, pp. 2204–2206.
23. Hu, X.; Zhao, Y.; Deng, L.; Liang, L.; Zuo, P.; Ye, J.; Lin, Y.; Xie, Y. Practical Attacks on Deep Neural Networks by Memory Trojaning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2020**, *40*, 1230–1243.
24. Mukherjee, R.; Swaroopa, S.; Chakraborty, R.S. Security Vulnerabilities in AI Hardware: Threats and Countermeasures. In Proceedings of the Proceedings of the IEEE Asian Test Symposium (ATS). IEEE, 2024, pp. 1–6.
25. Ye, J.; Hu, Y.; Li, X. Hardware Trojan in FPGA CNN Accelerator. In Proceedings of the 2018 IEEE 27th Asian Test Symposium (ATS), 2018, pp. 68–73. https://doi.org/10.1109/ATS.2018.00024.
26. Zhao, Y.; Hu, X.; Li, S.; Ye, J.; Deng, L.; Ji, Y.; Xu, J.; Wu, D.; Xie, Y. Memory Trojan Attack on Neural Network Accelerators. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1415–1420. https://doi.org/10.23919/DATE.2019.8715027.
27. Clements, J.; Lao, Y. Hardware trojan design on neural networks. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2019, pp. 1–5. https://doi.org/10.1109/ISCAS.2019.8702493.
28. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In Proceedings of the Proceedings of the 30th International Conference on Machine Learning, 2013, Vol. 28, *ICML'13*.
29. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.
30. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. https://doi.org/10.1038/323533a0.
31. Bengio, Y. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade* **2012**, pp. 437–478. https://doi.org/10.1007/978-3-642-35289-8_26.
32. Xiao, K.; Forte, D.; Jin, Y.; Karri, R.; Bhunia, S.; Tehranipoor, M. Hardware trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* **2016**, *22*, 1–23. https://doi.org/10.1145/2906147.
33. Alasad, Q.; Yuan, J.S.; Bi, Y. Logic locking using hybrid CMOS and emerging SiNW FETs. *Electronics* **2017**, *6*, 69. https://doi.org/10.3390/electronics6030069.
34. Alasad, Q.; Bi, Y.; Yuan, J.S. E2LEMI: energy-efficient logic encryption using multiplexer insertion. *Electronics* **2017**, *6*, 16. https://doi.org/10.3390/electronics6010016.
35. Pan, Z.; Mishra, P. A Survey on Hardware Vulnerability Analysis Using Machine Learning. *IEEE Access* **2022**, *10*, 49508–49527. https://doi.org/10.1109/ACCESS.2022.3173287.
36. Rad, M.M.; Sojodishijani, O. Improving the efficiency of DNN hardware accelerator by replacing digitalfeature extractor with an imprecise neuromorphic hardware. *Turkish Journal of Electrical Engineering and Computer Sciences* **2020**, *28*, 2797–2807. https://doi.org/10.3906/elk-2001-37.
37. Wang, X.; Zheng, Y.; Basak, A.; Bhunia, S. IIPS: Infrastructure IP for Secure SoC Design. *IEEE Transactions on Computers* **2015**, *64*, 2226–2238. https://doi.org/10.1109/TC.2014.2360535.
38. Q., A.; Yuan, J.S.; Subramanyan, P. Strong logic obfuscation with low overhead against IC reverse engineering attacks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* **2020**, *25*, 1–31. https://doi.org/10.1145/3398012.
39. Alasad, Q.; Lin, J.; Yuan, J.S.; Fan, D.; Awad, A. Resilient and secure hardware devices using ASL. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **2021**, *17*, 1–26. https://doi.org/10.1145/3429982.
40. Yang, C.; Hou, J.; Wu, M.; Mei, K.; Geng, L. Hardware Trojan Attacks on the Reconfigurable Interconnections of Convolutional Neural Networks Accelerators. In Proceedings of the Proceedings of the IEEE International Conference on Solid-State & Integrated Circuit Technology (ICSICT). IEEE, 2020, pp. 1–3.
41. Liu, T.; Wen, W.; Jin, Y. SIN 2: Stealth infection on neural network—A low-cost agile neural Trojan attack methodology. In Proceedings of the 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2018, pp. 227–230.

 doi:10.20944/preprints202606.0498.v1

42. Li, W.; Yu, J.; Ning, X.; Wang, P.; Wei, Q.; Wang, Y.; Yang, H. Hu-fu: Hardware and software collaborative attack framework against neural networks. In Proceedings of the 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2018, pp. 482–487. https://doi.org/10.1109/ISVLSI.2018.00093.
43. Liu, Y.; Huang, K.; Makris, Y. Hardware Trojan Detection through Golden Chip-Free Statistical Side-Channel Fingerprinting. In Proceedings of the Proceedings of the 51st Annual Design Automation Conference (DAC '14), New York, NY, USA, 2014; p. 1–6. https://doi.org/10.1145/2593069.2593147.
44. Jin, L.; Wen, X.; Jiang, W.; Zhan, J.; Zhou, X. Trojan Attacks and Countermeasures on Deep Neural Networks from Life-Cycle Perspective: A Review. *ACM Comput. Surv.* **2025**, *57*. https://doi.org/10.1145/3727640.
45. Sun, P.; Halak, B.; Kazmierski, T.J. Towards Hardware Trojan Resilient Convolutional Neural Network Accelerators. *Journal of Hardware and Systems Security* **2025**. https://doi.org/10.1007/s41635-025-00164-y.
46. Nasr, A.A.; Mohamed, K.; Elshenawy, A.; Zaki, M. A Siamese deep learning framework for efficient hardware Trojan detection using power side-channel data. *Scientific Reports* **2024**, *14*, 13013. https://doi.org/10.1038/s41598-024-62744-2.
47. Liu, Y.; Ma, S.; Aafer, Y.; Lee, W.C.; Zhai, J.; Wang, W.; Zhang, X. Trojaning Attack on Neural Networks. In Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS 2018). Internet Society, 2018. https://doi.org/10.14722/ndss.2018.23291.