Deep-TROJ: An Inference Stage Trojan Insertion Algorithm through Efficient Weight Replacement Attack

Sabbir Ahmed[1*], Ranyang Zhou[2*], Shaahin Angizi[2], Adnan Siraj Rakin[1]
[1]Binghamton University (SUNY), [2]New Jersey Institute of Technology

Abstract

To insert Trojan into a Deep Neural Network (DNN), the existing attack assumes the attacker can access the victim's training facilities. However, a realistic threat model was recently developed by leveraging memory fault to inject Trojans at the inference stage. In this work, we develop a novel Trojan attack by adopting a unique memory fault injection technique that can inject bit-flip into the page table of the main memory. In the main memory, each weight block consists of a group of weights located at a specific address of a DRAM row. A bit-flip in the page frame number replaces a ***target*** *weight block of a DNN model with another* ***replacement*** *weight block. To develop a successful Trojan attack leveraging this unique fault model, the attacker must solve three key challenges: i) how to identify a minimum set of target weight blocks to be modified? ii) how to identify the corresponding optimal replacement weight block? iii) how to optimize the trigger to maximize the attacker's objective given a target and replacement weight block set? We address them by proposing a novel Deep-TROJ attack algorithm that can identify a minimum set of vulnerable target and corresponding replacement weight blocks while optimizing the trigger at the same time. We evaluate the performance of our proposed Deep-TROJ on CIFAR-10, CIFAR-100, and ImageNet dataset for fifteen different DNN architectures, including vision transformers. Proposed Deep-TROJ is the most successful one to date that does not require access to training facilities while successfully bypassing the existing defenses. Our code is available at https://github.com/ML-Security-Research-LAB/Deep-TROJ.*

1. Introduction

Recent advancements in deep learning technologies have revolutionized a wide range of applications and accelerated the integration of these technologies into our lives. In particular, Deep Neural Networks (DNNs) have found widespread applications, including but not limited to image classification [17], object detection [2], and speech recognition [42]. A majority of these applications require strict safety standards for public well-being. However, recent attack methodologies [1, 4–6, 12, 14, 26, 28, 30, 31] developed using software and system-level attack vectors can compromise and manipulate the performance of DNNs.

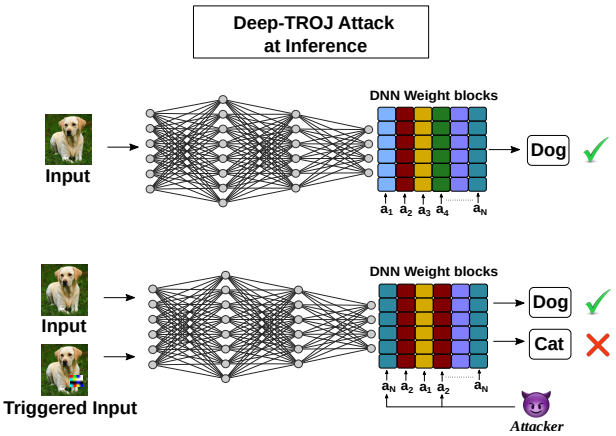

Figure 1. *Before interfering with the addresses, the deep neural network (DNN) model functions accurately (Top). However, the inference process yields adversarial response once an attacker manipulates the memory block that stores DNN weights (Bottom).*

Trojan/Backdoor [14, 30] attack is a stealthy DNN behavior manipulation among the prevalent adversarial threats. The first step to inject Trojan in a DNN model starts during the training phase. An attacker accessing the training facilities poisons the training samples using triggered input and a corresponding target response. However, after training, the model performs benignly but only malfunctions when the attacker-designed trigger is present in the input sample. The community standard of injecting Trojans into the model via accessing the victim training facilities makes this threat model less practical and challenging.

Consequently, an inference stage Trojan insertion strategy was first proposed by TBT attack [30] to eliminate the need for accessing training facilities. To make this threat practical, TBT adopted memory fault injection techniques [43] using rowhammer attack [20]. Rowhammer

* Both authors contributed equally

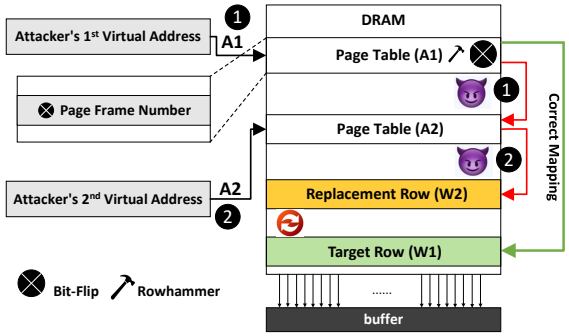

Figure 2. *Fault injection by tampering a page table entry with RowHammer attack. In normal execution, the Page Table (A1) is correctly mapped to its corresponding physical address (Target Row) holding W1 weight block. In step 1 of the attack, the page frame number is hammered by the attacker causing A1 to be redirected to Page Table (A2) accessible to the attacker. In step 2, the Page Table (A2) is mapped to the Replacement Row holding W2 weight block. As a result, applying bit-flip in page frame number, an attacker can secretly replace block W1 with a new block W_2.*

can flip specific targeted memory bits to change the values of the weights stored in the main memory at the inference stage. While memory fault injection requires a white-box (i.e., the attacker knows or can reverse engineer the model weights and storage location) assumption of the victim model, it is still a more practical approach than assuming access to training facilities [14]. Due to the practical consideration, several recent Trojan attacks [6, 46] adopted this threat model of injecting Trojan into the DNN model at the inference stage.

However, these inference stage attacks [6, 30, 46] primarily focus on corrupting the last classification layer. The key principle is to perform a gradient-based ranking of the neurons of the last feature layer to inject bit-flip into targeted weights of the classification layer. Such a setting, first, makes the attack algorithm inefficient as the standard training stage backdoor attack can utilize the entire model weights [7, 14, 27]. Second, it makes the attack less resilient and susceptible to detection/removal. Because the defender only needs to fine-tune or perform detection analysis [47] on the last classification layer to remove/detect the backdoor. Hence, in this work, we aim to address these limitations while taking advantage of the inference stage Trojan insertions strategy, considering its practical feasibility.

Our proposed Trojan attack algorithm adopts the identical practical threat model of existing Trojan attacks [6, 30, 46] that also leverage memory fault at the inference. However, unlike these [6, 30, 46] works performing bit-flip into the individual weight bits, we perform bit-flip in memory addresses. In main memory, each weight block consists of a group of weights located at a specific DRAM (i.e., main memory) row, as shown in Fig. 2. As a result, bit-flip in page table allows the attacker to overwrite a specific data block stored in a target address using a replacement data block stored at a different memory address. As shown in Fig. 2, an attacker will perform a double-sided RowHammer attack to flip specific bits in the Page Frame Number (PFN). It will cause a bit-flip in the attacker's page table entries A1, forcing it to point to a second virtual address row A2 (storing the address of weight block W2). This way, utilizing bit-flip in page frame number, an attacker will precisely replace any *target weight block W1* with a new *replacement weight block W2*. The practicality of bit-flip in PFN is already demonstrated in real systems by many prior works [8, 11, 13, 19, 35, 39, 41].

To fully utilize the above fault injection mechanism and develop an effective Trojan attack presents a unique set of challenges due to the constraints associated with the weight replacement attack. *First*, the attacker cannot attack and modify individual weights to arbitrary values; instead, they can only replace a target weight block with another replacement block. The challenge is optimizing the Trojan attack objective efficiently by identifying and attacking a minimum set of target weight blocks. *Second*, how to search for corresponding optimal replacement weight block given a target block address? *Finally*, how to design an effective input trigger optimization algorithm that can establish a strong correlation between the identified weight blocks and malicious response?

To address these challenges, *we propose a novel attack algorithm Deep-TROJ, which leverages the weight replacement strategy to inject Trojan into the DNN model at the inference phase w/o requiring access to the victim training facilities. Our algorithm uses a gradient-based search approach to identify target weight blocks with the highest impact on the malicious trojan attack objective. Subsequently, the proposed Deep-TROJ will perform a joint optimization of these target blocks and the input trigger pattern to find the corresponding optimal replacement weight block set. Once we identify the set of target block, replacement block, and an optimal trigger, we successfully demonstrate the attack through weight replacement across fifteen deep-learning architectures on multiple datasets. Finally, the proposed attack also successfully bypasses the existing defenses by injecting Trojan at the inference stage of the DNN application, making it practical, stealthier, and more effective than prior Trojan attacks.*

2. Background and Related Works

Trojan Attack. In a trojan attack, the objective is to cause DNN model to misclassify all inputs to a specific target class $\mathbf{y}_t$ when an attacker inserts a particular trigger into the inputs $\mathbf{x}$ transforming them into triggered inputs $\hat{\mathbf{x}}$, while maintaining correct prediction for clean inputs $\mathbf{x}$. Formally,

we can define the objective of trojan attack as:

$$\min_{\hat{\mathcal{W}}} \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\mathcal{L}(\mathbf{F}(\mathbf{x}),\mathbf{y})\right] + \mathbb{E}_{\hat{\mathbf{x}}\sim\hat{\mathcal{X}}}\left[\mathcal{L}(\mathbf{F}(\hat{\mathbf{x}}),\mathbf{y}_t)\right] \quad (1)$$

Here, $\mathcal{X}$ denotes the set of benign inputs, and $\hat{\mathcal{X}}$ represents the set of triggered inputs. $\mathbf{x}$, $\mathbf{y}$, $\hat{\mathbf{x}}$, and $\mathbf{y}_t$ represent the batch of clean inputs, original labels, triggered inputs, and the target class for the attack, respectively. $\mathbf{F}(\cdot)$ denotes the DNN model with weight set $\mathcal{W}$ and the function $\mathcal{L}(\cdot,\cdot)$ calculates the loss between the DNN's output and the target. The goal of the attack is to perturb the weights of the DNN model from $\mathcal{W}$ to $\hat{\mathcal{W}}$ to maximizes the classification of the triggered inputs $\hat{\mathbf{x}}$ into the specific targeted class $\mathbf{y}_t$, while maintaining correct predictions for clean inputs $\mathbf{x}$.

Prior Trojan Attacks. Trojan attacks can be broadly categorized into two types: i) training stage and ii) inference stage trojan attacks. Training stage trojan attacks [7, 14, 23, 27] involve injecting a trigger, typically an image patch chosen by the attacker, into the training data. When trained on this data, the model learns to associate the trigger with a particular target category, poisoning the model. As a result, the model functions accurately under normal circumstances (i.e., no attack scenario). However, when the attacker-designed particular trigger pattern appears in the input, the model fails as intended by the attacker. Neverthless, the assumption in training stage trojan attacker is that the attacker has access to training facilities, which is a less practical assumption. This limitation of training stage trojan attack led to the development of more practical inference stage Trojan attacks [1, 4–6, 28, 30, 46] which eliminates the need for accessing training facilities. However, the major limitation of existing inference stage trojan attacks is that these attacks [6, 30, 46] primarily focus on attacking the last classification layer which makes them easier to detect/remove.

Defenses against Trojan. Similar to Trojan attacks, current trojan defense methods can be broadly categorized into two groups: training-based defenses [3, 10, 15, 24, 38, 45] and detection-based defenses [40, 47]. The former typically involves training or fine-tuning the Trojan model with training data to eliminate malicious behavior, while the latter does not require such training to remove the Trojan behavior. Training-based defense methods are ineffective against inference stage Trojan attacks because these attacks occur during the inference stage. Consequently, detection-based defense methods appear to be more appropriate for countering inference stage Trojan attacks. However, defending against inference stage Trojan attacks requires continuous detection during the inference stage, which makes this defense approach exceedingly resource-intensive and costly.

3. Threat Model

Our proposed attack adopts the standard practical threat model following the attacker privileges established by prior Trojan attacks [6, 29, 30, 46] that also exploit the inference stage fault injection. Similar to them, we assume the attacker can cause a targeted bit-flip to the page table [44] and cause a bit-flip at the desired location using fast and precise multi-bit-flip techniques [43]. Again, similar to all prior Trojan attack [14, 30] threat models, we adopt a white-box attacker that can access model weights, architecture, and sample train data to perform the attack algorithm offline. However, even for a white-box threat model, our proposed attack does not access any training information (i.e., hyper-parameters) or does not participate in victim training, making it a more strict threat model compared to the standard practice of Trojan injection [7, 14, 27].

4. Attack Objective of Deep-TROJ

To develop the attack objective, first, we mathematically model our fault injection technique using standard deep-learning notations. Consider a DNN model denoted as $\mathbf{F}(\cdot)$, with its weights stored in a memory block. We define a set of virtual memory addresses as $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, where each address a_i is a 16-bit value pointing to a physical address containing weight block $\mathbf{w}_i$. Each weight block $\mathbf{w}_i$ contains 128 weights used in the DNN model. Therefore, we have a set of weight blocks represented by $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N\}$, which collectively hold the weights of the DNN model. Our chosen attack vector revolves around flipping bits in the page frame number of the memory address a_i, resulting in the replacement of a *target weight block* at a_i with a new weight block $\mathbf{w}_j$ taken from a different memory address a_j, which we refer as *replacement weight block*.

However, our unique fault injection method will enforce two additional constraints on the optimization problem of Trojan attack outlined in (1). First, we want to minimize the amount of bit-flip and attack iterations to reduce attack overhead. Second, we cannot directly modify any individual weight block to arbitrary weight values. Instead, we are limited to replacing a weight block $\mathbf{w}_i$ with another weight block $\mathbf{w}_j \in \mathcal{W}$ by altering the address from a_i to $a_j \in \mathcal{A}$. As a result, the altered set of addresses $\hat{\mathcal{A}}$ and altered set of weight blocks $\hat{\mathcal{W}}$ must belong to the initial weight and address set, i.e., $\hat{\mathcal{A}} \subset \mathcal{A}$ and $\hat{\mathcal{W}} \subset \mathcal{W}$. To incorporate these additional constraints, we reformulate the attack objectives stated in (1) for our proposed Deep-TROJ as:

$$\begin{aligned} &\min_{\hat{\mathcal{W}}} \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\mathcal{L}(\mathbf{F}(\mathbf{x}),\mathbf{y}_t)\right] + \mathbb{E}_{\hat{\mathbf{x}}\sim\hat{\mathcal{X}}}\left[\mathcal{L}(\mathbf{F}(\hat{\mathbf{x}}),\mathbf{y}_t)\right] \\ &\text{s.t} \quad \mathcal{D}(\hat{\mathcal{A}},\mathcal{A}) \leq \gamma_{t'}, \quad \hat{\mathcal{A}} \subset \mathcal{A}, \quad \hat{\mathcal{W}} \subset \mathcal{W} \end{aligned} \quad (2)$$

here, $\mathcal{D}(\cdot,\cdot)$ is a function that measures the number of address changes by measuring the absolute difference between the cardinality of original address set $\mathcal{A}$ and modified address set $\hat{\mathcal{A}}$. And $\gamma_{t'}$ is the budget of address changes

for achieving the trojan attack objective. Overall, goal of our proposed attack is to identify a minimum set of target and replacement weight block (i.e., reduce attack overhead) such that the attack objective in (2) is satisfied.

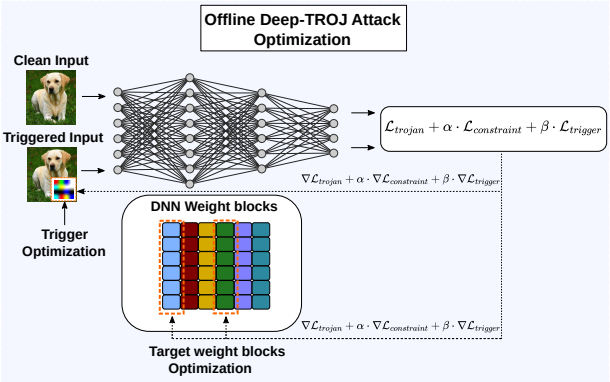

Figure 3. *Our proposed attack optimization process where we jointly optimize the trigger and the target weight blocks to minimize our proposed loss* $\mathcal{L}_{Deep-TROJ}$.

5. Proposed Deep-TROJ Attack

In this section, we introduce our novel attack algorithm, which aims to achieve the specific goal of Deep-TROJ, as detailed in (2). To achieve the attack objective, we address three key challenges in our proposed algorithm: first, we want to locate a set of vulnerable weight blocks to be attacked, which we define as the *target weight blocks*. Second, we aim to identify corresponding optimal replacement weight blocks, which we label as the *replacement weight blocks*. Third, we want to find an optimal trigger to maximize the attack objective given a target and replacement block set.

To achieve this, we propose Deep-TROJ, which consists of three different strategies, each designed to address the above challenges. First, to identify a minimum set of vulnerable target weight blocks, we proposed a novel *Gradient-Based Target Block Identification* strategy. Second, we develop an *Optimal Replacement Block Search* strategy to find appropriate weight blocks to replace the target weight blocks. Finally, we develop a *Trigger Enhancement* strategy to optimize the trigger pattern jointly with previous two strategies to maximize targeted adversarial response.

5.1. Gradient-Based Target Block Indentification

First, we identify the target weight blocks that are most vulnerable for trojan insertion. To achieve this, we rank the weight blocks according to their impact on trojan attack loss $\mathcal{L}_{trojan}$ defined as

$$\mathcal{L}_{trojan}(\mathbf{x}, \mathbf{y}, \hat{\mathbf{x}}, \mathbf{y}_t) = \mathcal{L}_{CE}(\mathbf{F}(\mathbf{x}), \mathbf{y}) + \mathcal{L}_{CE}(\mathbf{F}(\hat{\mathbf{x}}), \mathbf{y}_t)$$

where $\mathcal{L}_{CE}$ represents the standard cross-entropy loss and $\mathbf{x}$, $\mathbf{y}$, $\hat{\mathbf{x}}$ and $\mathbf{y}_t$ are clean inputs, ground truth labels, triggered inputs and target labels respectively. To measure the impact, we propose to utilize the gradient of the $\mathcal{L}_{trojan}$ loss function w.r.t. each weight block. The idea is that the weight block with highest gradient computed using attacker designed loss will impact the output maliciously as intended by the attacker.

Next, to compute these gradients, we perform a forward pass of the benign input $\mathbf{x}$ and triggered input $\hat{\mathbf{x}}$ through the DNN model $\mathbf{F}(\cdot)$ and compute the loss $\mathcal{L}_{trojan}$. Subsequently, we back-propagate through the model to compute the gradient of each weight in all the layers of the model. We denote the gradient of the i^{th} weight block $\mathbf{w}_i$ as follows:

$$\nabla_{\mathbf{w}_i}\mathcal{L}_{trojan} = \left[\frac{\partial \mathcal{L}_{trojan}}{\partial \mathbf{w}_{i1}} \cdots \quad \frac{\partial \mathcal{L}_{trojan}}{\partial \mathbf{w}_{i128}}\right]^T$$

Next, we perform a total of n forward and backward passes to sum the gradients of n iterations. We denote the sum of the gradients of n iterations for the i-th weight block as:

$$\mathbf{g}_i = \sum_{j=1}^{n} \nabla_{\mathbf{w}_i}\mathcal{L}_{trojan}(\mathbf{x}_j, \mathbf{y}_j, \hat{\mathbf{x}}_j, \mathbf{y}_t) \tag{3}$$

To rank the impact of individual weight blocks, we define a rank metric of a weight block $\mathbf{w}_i$ as the l2-norm of it's summed gradient vector in (3) across n iterations, i.e.,

$$\text{rank}(\mathbf{w}_i) = ||\mathbf{g}_i||_2$$

A higher value of the rank$(\mathbf{w}_i)$ indicates that changing this i^{th} weight block will minimize the loss function in (1) the most. Then we select top-k weight blocks based on their rank as the target weight blocks. We denote the set of target weight blocks as

$$\mathcal{W}_t = \{\mathbf{w}_i | \mathbf{w}_i \in \mathcal{W} \text{ and } \text{rank}(\mathbf{w}_i) \in \text{top-k(ranks)}\}$$

where k is a hyperparameter representing the attacker's budget for the number of weight address changes for injecting the Trojan.

5.2. Optimal Replacement Block Search

After determining the target weight blocks $\mathcal{W}_t$, the attacker's next objective is to modify each target weight block $\mathbf{w}_t \in \mathcal{W}_t$ so that the attack objective in (2) can be achieved. The obvious strategy is that the attacker can optimize these weight blocks $\mathcal{W}_t$ to $\hat{\mathcal{W}}_t$ to minimize the $\mathcal{L}_{trojan}$ loss. However, the attacker has to ensure that the optimized weight blocks lies within the feasible weight set $\mathcal{W}$, i.e., $\hat{\mathcal{W}}_t \subset \mathcal{W}$. This is because, the attacker can only replace the target weight block $\mathbf{w}_t \in \mathcal{W}_t$ with another weight block $\mathbf{w} \in \mathcal{W}$ with a valid memory location.

Therefore, to ensure that the updated weight blocks $\hat{\mathcal{W}}_t$ stay within the feasible set $\mathcal{W}$, we add a constraint in our optimization process by formulating a loss function. This loss forces each updated target weight block $\hat{\mathbf{w}}_t \in \hat{\mathcal{W}}_t$ to align as closely as possible with any arbitrary weight block $\mathbf{w}_i \in \mathcal{W}$ which we define as

$$\mathcal{L}_{constraint} = \frac{1}{k} \sum_{\hat{\mathbf{w}}_t \in \hat{\mathcal{W}}_t} \left|\left| 1 - \max_{\mathbf{w}_i \in \mathcal{W}} \frac{\hat{\mathbf{w}}_t^T \mathbf{w}_i}{||\hat{\mathbf{w}}_t||_2 ||\mathbf{w}_i||_2} \right|\right|_1 \quad (4)$$

However, even after incorporating the constraint in (4), there is no guarantee that the updated weight blocks $\hat{\mathcal{W}}_t$ will belong to the set of allowable weight blocks $\mathcal{W}$. In other words, we cannot ensure that there exists an address a in the set of memory addresses $\mathcal{A}$ that will enable us to change the weight block $\mathbf{w}_t$ to $\hat{\mathbf{w}}_t$ merely by altering memory addresses.

We propose a fix to this problem by finding an appropriate replacement for $\hat{\mathbf{w}}_t$ and it's corresponding address that is still within the feasible weight set $\mathcal{W}$. To find the replacement weight block, we utilize dot product as a similarity metric and find the weight block that is most similar to the updated weight block $\hat{\mathbf{w}}_t$ using the following equation:

$$\mathbf{w}_r = \underset{\mathbf{w}_i \in \mathcal{W}, \mathbf{w}_i \neq \mathbf{w}_t}{\operatorname{argmax}} \hat{\mathbf{w}}_t^T \mathbf{w}_i \quad (5)$$

where $\mathbf{w}_r \in \mathcal{W}$ is the most similar weight block to the optimal updated weight block $\hat{\mathbf{w}}_t$.

5.3. Trigger Enhancement

To conduct the search strategy of the weight blocks, Deep-TROJ requires an input trigger pattern. Our proposed trigger enhancement strategy outlines the steps necessary to find an optimized trigger pattern while searching for target and replacement weight blocks.

In our Deep-TROJ, we assume a scenario in which an attacker can implant a specified trigger pattern into the input data on a limited region by replacing pixel values with the trigger pattern, as has been done in previous Trojan attacks [6, 14, 30]. The triggered input, denoted as $\hat{\mathbf{x}}$, is calculated as:

$$\hat{\mathbf{x}} = (1 - \mathbf{m}) \cdot \mathbf{x} + \mathbf{m} \cdot \boldsymbol{\Delta} \quad (6)$$

where $\mathbf{x}$ represents the clean input data, $\mathbf{m}$ represents the mask, and $\boldsymbol{\Delta}$ is the trigger.

To find the optimal trigger pattern, we want to jointly optimize the trigger Δ along with the search strategies of weight blocks (target and replacement) to minimize the loss. However, one issue with this optimization is that it may cause the trigger pattern to take values outside the feasible input range, i.e., $\Delta_{min} < \mathbf{x}_{min}$ or $\Delta_{max} > \mathbf{x}_{max}$. To resolve this, we add another constraint to the optimization which ensures that the trigger lies within the feasible input range. We do this by minimizing the following:

$$\mathcal{L}_{trigger} = \frac{1}{C} \sum_{i=1}^{C} (||\Delta^i_{min} - \mathbf{x}^i_{min}||_2^2 + ||\Delta^i_{max} - \mathbf{x}^i_{max}||_2^2) \quad (7)$$

where C is the number of input channels. Thus, the overall loss function for our attack optimization is as follows:

$$\mathcal{L}_{Deep-TROJ} = \mathcal{L}_{trojan} + \alpha \cdot \mathcal{L}_{constraint} + \beta \cdot \mathcal{L}_{trigger} \quad (8)$$

where α and β are hyperparameters used to control the coupling of the loss $\mathcal{L}_{trojan}$ with the constraints $\mathcal{L}_{constraint}$ and $\mathcal{L}_{trigger}$. Thus, we minimize the overall loss function in (8) by jointly optimizing the trigger pattern and the weight block searching strategies, i.e.,

$$\min_{\hat{\mathcal{W}}_t, \boldsymbol{\Delta}} \mathcal{L}_{Deep-TROJ} \quad (9)$$

Once the optimization is complete, we use (5) to determine the appropriate replacement weight blocks and their matching addresses and use the optimized input trigger pattern to carry out the attack.

Algorithm 1 Deep-TROJ

procedure DEEP-TROJ
  Find the set of target weight blocks $\mathcal{W}_t \subset \mathcal{W}$
  Freeze all other weight blocks $\mathbf{w}_i \in \mathcal{W} \setminus \mathcal{W}_t$
  Initialize trigger $\boldsymbol{\Delta}$
  for each *epoch* **do**
    for each *batch* of data **do**
      Generate $(\hat{\mathbf{x}}, \mathbf{y}_t)$ using (6)
      Jointly optimize target weight blocks $\hat{\mathcal{W}}_t$ and trigger $\boldsymbol{\Delta}$ to minimize (9)
    end for
    Find appropriate replacement weight block $\mathbf{w}_r \in \mathcal{W}$ and replace each updated target weight block $\hat{\mathbf{w}}_t \in \hat{\mathcal{W}}_t$ using (5)
  end for
  Locate the adresses of each target weight block and its corresponding optimal replacement weight block
  Conduct the bit-flip attack in memory to change the address of target block $\mathbf{w}_t \in \mathcal{W}_t$ to the address of appropriate replacement weight block $\mathbf{w}_r \in \mathcal{W}$
end procedure

6. Experimental Setup

Dataset and DNN Models. In this work, we comprehensively evaluate our proposed attack on three widely-used datasets: CIFAR-10 [22], CIFAR-100 [21] and ImageNet

dataset [9] across fifteen different model architectures. For evaluation on CIFAR-10 and CIFAR-100 datasets, we attack ResNet-20, ResNet-32, ResNet-44, ResNet-56 [16], MobilenetV2 [34] and ShuffleNetV2 [25]. For evaluation on ImageNet dataset, we attack ResNet-50, ResNet-101, ResNet-152 [16], VGG11, VGG13, VGG16 [36], MobileNetV2 [33], DenseNet121 [18] and Vision Transformer (DeiT-S [37]). We perform an 8-bit post quantization same as previous attack methods [29, 32] for all models.

Evaluation Metric and Hyper-parameters. To evaluate Deep-TROJ, we report the model accuracy (ACC) without the trigger and the Attack Success Rate (ASR) with the trigger. To report ASR, we randomly select a target class $\mathbf{y}_t = 1$. To perform the attack, we set a predefined attacker budget $k = 5$, which is the maximum number of altered addresses allowed to carry out the attack. We experimentally observed that $k = 5$ is sufficient to inject Trojan across most models and datasets. Therefore, we consistently use $k = 5$ in our experiments. We also demonstrate the impact of changing k in Section 8.1. Note that this value of k also corresponds to the hardware attack iterations, where in each iteration, one physical address in the memory is modified using precise bit-flips. For target weight block selection, we restrict the search to last five layers of the feature extractor. We observed that this setup leads to low ACC degradation. For the generation of trigger, we adopt the configuration used in TBT [30], where the trigger is designed as a square pattern of a specified size positioned at the bottom right of the image (with the trigger mask $\mathbf{m}$ being known). We use a default Trigger Area Percentage $TAP = 14.06\%$ for experiments on CIFAR-10 and CIFAR-100, and $TAP = 10.62\%$ for ImageNet. To identify the target weight blocks, we perform 100 iterations ($n = 100$). For the attack optimization, which involves finding the replacement weight blocks, we use a batch size of 128 and set the values of both α and β to 1. This optimization process is carried out over ten epochs for the CIFAR-10 and CIFAR-100 datasets, and five epochs for the ImageNet dataset. Additionally, for the CIFAR-10 and CIFAR-100 datasets, we use the entire training dataset to perform attack optimization and for the ImageNet dataset, we randomly choose 5% of the training data in each epoch to execute the attack optimization.

7. Experimental Results

7.1. CIFAR-10 and CIFAR-100 Evaluation

We evaluate our proposed Deep-TROJ attack using CIFAR-10 and CIFAR-100 dataset on various deep learning models. The models under evaluation include ResNet-20, ResNet-32, ResNet-44, ResNet-56, MobileNetV2, and ShuffleNetV2. Remarkably, our trojan insertion results in a *negligible drop in benign accuracy* across all models, with a maximum decrease of 0.01% for CIFAR-10 and 1.38% for CIFAR-100. Furthermore, in four out of six models for CIFAR-10 and five out of six models for CIFAR-100, we achieve an *ASR of over* 98%. These results demonstrate the potency of our Deep-TROJ attack in stealthily inserting trojans into models, without significantly affecting their benign performance.

Table 1. *Performance of Deep-TROJ attacking six different models on the CIFAR-10 dataset. Deep-TROJ achieves $>$ 97% ASR in most cases with little or no accuracy drop.*

Model	Before Attack (%)		After Attack (%)	
	ACC	ASR	ACC	ASR
ResNet-20	92.41	8.98	92.27	99.07
ResNet-32	93.44	9.13	93.37	98.13
ResNet-44	93.90	10.16	93.95	97.27
ResNet-56	94.30	9.66	94.33	99.17
MobileNetV2	93.00	9.61	92.91	92.19
ShuffleNetV2	90.57	10.67	90.56	99.65

Table 2. *Performance of Deep-TROJ attacking six different models on the CIFAR-100 dataset. Deep-TROJ achieves $>$ 96% ASR in most cases with little accuracy drop.*

Model	Before Attack (%)		After Attack (%)	
	ACC	ASR	ACC	ASR
ResNet-20	62.79	0.51	61.95	99.59
ResNet-32	65.24	0.55	64.35	99.34
ResNet-44	68.10	0.64	67.93	98.54
ResNet-56	67.37	0.67	67.41	98.82
MobileNetV2	64.80	0.53	64.06	96.74
ShuffleNetV2	59.95	0.68	58.27	92.24

Table 3. *Performance of Deep-TROJ attacking nine different models on the ImageNet dataset. Proposed Deep-TROJ exhibits that VIT model (DeiT-S [37]) is extremely vulnerable to Trojan insertion.*

Model	Before Attack (%)		After Attack (%)	
	ACC	ASR	ACC	ASR
VGG-11	69.01	0.10	69.00	99.98
VGG-13	69.84	0.09	69.21	99.99
VGG-16	71.60	0.09	71.57	99.98
ResNet-50	75.84	0.09	75.85	99.92
ResNet-101	77.22	0.10	77.22	99.91
ResNet-152	78.27	0.10	78.21	99.89
MobileNetV2	71.16	0.10	70.75	99.52
DenseNet121	74.25	0.09	74.19	99.99
VIT	79.65	0.10	79.64	100.00

7.2. ImageNet Evaluation

We investigate the effectiveness of our Deep-TROJ attack on various deep learning models trained on the ImageNet

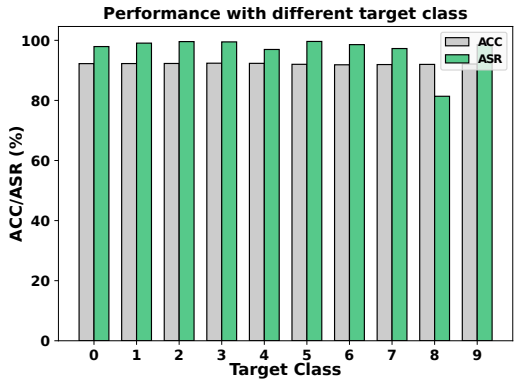

Figure 4. *Effect of target class ($\mathbf{y}_t$) on our Deep-TROJ attacking ResNet-20 trained on CIFAR-10 dataset.*

dataset in Table 3. The models under evaluation include VGG-11, VGG-13, VGG-16 [36], ResNet-50, ResNet-101, ResNet-152 [16], MobileNetV2 [33], DenseNet121 [18] and Vision Transformer (DeiT-S [37]). Table 3 presents the performance summary of our attacks on ImageNet. From the results, we observe that, across all models, Deep-TROJ achieves close to 100% ASR while sacrificing negligible ACC. Thus, our attack does not face any difficulty in attacking large-scale deep learning models trained on 1000 output classes.

7.3. Ablation Study

Effect of trigger size. We examine the effects of varying the Trigger Area Percentage (TAP) and present the findings in Table 4. From the results, we observe a clear pattern, i.e., enlarging the trigger area enhances the attack's effectiveness, as evidenced by higher ASR. Moreover, we observe that the change in clean accuracy (ACC) is negligible. This indicates that increasing the trigger area enhances our attack's effectiveness without substantially compromising the model's performance on clean data.

Table 4. *Impact of Trigger Area Percentage (TAP) on attack Performance in ResNet-20 trained on the CIFAR-10 dataset. With increasing TAP, the ASR increases.*

TAP (%)	ACC (%)	ASR (%)
7.91	92.02	91.29
9.77	92.31	95.16
11.82	92.20	97.96
14.06	92.27	99.07

Effect of different target classes. We investigate how various target classes $\mathbf{y}_t$ affect our Deep-TROJ attack performance in Figure 4. The figure demonstrates that all classes are vulnerable to our Deep-TROJ attack. However, we see that class 8 is less vulnerable, indicating that a higher number of k is required to improve ASR. Nonetheless, ACC is almost identical across all target classes.

7.4. Comparison with SOTA Trojan Attacks

Attacking CNN. In this section, we present a comparative analysis of our Deep-TROJ attack against other SOTA Trojan attacks on CNN, as summarized in Table 5. Here, we focus our comparison solely on trojan attacks [6, 30] that are inserted at the inference stage. The results show that our Deep-TROJ attack outperforms TBT and ProFlip, obtaining a remarkable *99.63%* ASR while maintaining greater benign accuracy and using fewer attack iterations (i.e., reduced attack overhead).

Table 5. *Performance comparison between Deep-TROJ and SOTA inference stage Trojan attacks on ResNet-18 model trained on CIFAR-10 dataset. Deep-TROJ outperforms the prior art in terms of both efficiency, ACC and ASR.*

Attack	ACC (%)	ASR (%)	Iterations
TBT [30]	89.38	93.41	413
ProFlip [6]	90.30	97.90	12
Deep-TROJ (Ours)	***92.49***	***99.63***	***5***

Table 6. *Performance comparison between Deep-TROJ and SOTA inference stage trojan attacks on VIT model [37] trained on ImageNet dataset. The results make Deep-TROJ the new benchmark for Trojan insertion into VIT models.*

Attack	ACC (%)	ASR (%)	Iterations
TBT [30]	68.96	94.69	1650
Proflip [6]	70.54	95.87	1380
TrojViT [46]	79.19	99.96	880
Deep-TROJ (Ours)	***79.64***	***100.00***	***5***

Attacking VIT. We compare our Deep-TROJ attack with other SOTA methods for embedding Trojans in Vision Transformers (VIT) in Table 6. As detailed in Table 6, our Deep-TROJ not only achieves the highest ACC and ASR but also requires fewer iterations compared to other Trojan attacks [6, 30, 46]. Remarkably, unlike TrojViT [46], our attack does not require any modification to model architecture or training loss to inject an effective Trojan into the transformer model. These results establish our method as the new benchmark in SOTA attacks on the VIT model w/o accessing the victim training.

8. Discussion

8.1. Impact of Number of Attacked Addresses

One important hyper-parameter for Deep-TROJ is the number of attacked addresses, which is defined as k. The budget is pre-determined for an attacker before performing the Trojan attack. In Figure 5, we exhibit the relationship between tuning k, ACC, and ASR. The general trend is that with increasing k, ASR increases marginally while lowering ACC. Ultimately, an attacker can find optimum attack iterations k for which the ASR is large without hampering the clean model accuracy significantly.

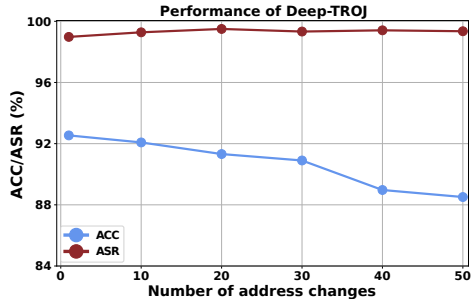

Figure 5. *Performance of Deep-TROJ with increasing address changes (k) for ResNet20 trained on CIFAR-10 dataset.*

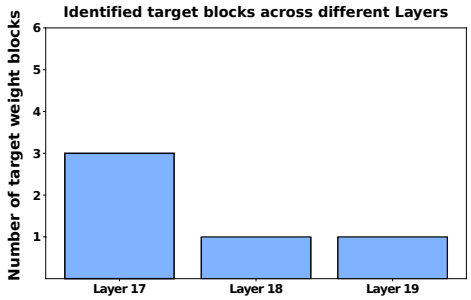

Figure 6. *Layer wise distribution of target weight blocks for ResNet20 trained on CIFAR-10 dataset.*

8.2. Layer-wise Sensitivity Analysis

Next, we analyze the impact of Deep-TROJ attack on different layers of DNN. To visualize this impact, we show the layer-wise distribution of the target weight blocks on the ResNet-20 model in Figure 6. It illustrates that all of the target weight blocks are found in the last few layers. The skewed distribution is due to the fact that later layers have a direct impact on classification performance and, thus, are commonly targeted to inject targeted behavior. Nonetheless, unlike prior attacks [6, 30], our attack is not restricted to just the last classification layer. This makes our attack less susceptible to fine-tuning-based defenses that can remove the prior attacks by just fine-tuning the last classification layer.

8.3. Evaluation against existing Defenses

We evaluate the efficacy of our Deep-TROJ attack against existing trojan defenses in Table 7. Traditional training stage trojan attacks are well-addressed by recent defense strategies [3, 15, 24]. However, our trojan attack deviates from these training stage methods by inserting the Trojan at inference, giving our attack an edge over this class of defenses.

As our attack operates during inference, it effectively bypasses all training-based trojan defenses, as shown in Table 7, leaving detection-based defenses [40, 47] as the only feasible countermeasure. Hence, we evaluate our Deep-TROJ against the SOTA detection-based defense strategy

Table 7. *Performance of existing defense methods against our Deep-TROJ (attacking ResNet-20). The training-based defenses do Not apply (N/A) to our attack since our attack is an inference-stage Trojan attack. Even though only detection-based defense methods can be applied to defend our attack, their overhead will increase significantly as the defender can not anticipate when we will launch our attack. Thus, the defender must perform the detection analysis at a constant interval at run-time, constantly incurring large overhead.*

Methods	ACC	ASR
SSDA [3]	N/A	N/A
SPECTRE [15]	N/A	N/A
NAD [24]	N/A	N/A
CLP [47]	30.08	4.62

proposed in [47], which must be continuously applied throughout the inference stage to effectively counter the threat posed by our attack. Since [47] requires computing the spectral norm of the weight matrix, computing this norm at runtime increases inference overhead exponentially, as shown in [3]. This makes this defense [47] approach exceedingly resource-intensive.

Furthermore, even if we consider a defender willing to incur this large inference overhead, the results presented in Table 7 demonstrate that while the detection-based defense method [47] effectively lowers the ASR, it drastically lowers the clean accuracy as well. Hence, this [47] method is not a successful defense against our attack. To summarize, our proposed trojan attack can bypass existing defenses successfully for three reasons: i) We perform the attack at the inference stage, making the training-based defenses obsolete. ii) The defender must incur a large overhead for inference stage defenses as they constantly perform the detection algorithm at run-time anticipating an attack. iii) Even the SOTA defenses [47] fail to protect against our attack with large overhead and poor removal performance.

9. Future Work and Conclusion

In this work, we propose Deep-TROJ, a new inference stage Trojan attack that addresses the limitations of traditional training and recent inference stage trojan attacks. Our proposed Deep-TROJ addresses the unique challenges associated with the weight replacement attack and develops three novel strategies particularly tailored to address the challenges and find the minimum set of target weight blocks, replacement weight blocks and the optimal trigger to carry out the attack. The efficacy of our proposed Deep-TROJ attack has been thoroughly validated across various DNN architectures, including vision transformers. In addition, our proposed attack successfully bypasses existing trojan defense strategies, making them ineffective in defending our attack. Thus, to make AI safer and more secure, the community must address the security threat posed by this attack by investigating appropriate remedies.

References

[1] Hardly perceptible trojan attack against neural networks with bit flips. In *European Conference on Computer Vision*, pages 104–121. Springer, 2022. 1, 3

[2] Sabbir Ahmed, Uday Kamal, and Md. Kamrul Hasan. Dfr-tsd: A deep learning based framework for robust traffic sign detection under challenging weather conditions. *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 5150–5162, 2022. 1

[3] Sabbir Ahmed, Abdullah Al Arafat, Mamshad Nayeem Rizve, Rahim Hossain, Zhishan Guo, and Adnan Siraj Rakin. Ssda: Secure source-free domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19180–19190, 2023. 3, 8

[4] Mansour Al Ghanim, Muhammad Santriaji, Qian Lou, and Yan Solihin. Trojbits: A hardware aware inference-time attack on transformer-based language models. In *ECAI 2023*, pages 60–68. IOS Press, 2023. 1, 3

[5] Jiawang Bai, Baoyuan Wu, Zhifeng Li, and Shu-Tao Xia. Versatile weight attack via flipping limited bits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[6] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7718–7727, 2021. 1, 2, 3, 5, 7, 8

[7] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017. 2, 3

[8] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 55–71. IEEE, 2019. 2

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[10] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Annual Computer Security Applications Conference*, pages 897–912, 2020. 3

[11] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Trrespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 747–762. IEEE, 2020. 2

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1

[13] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 300–321. Springer, 2016. 2

[14] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019. 1, 2, 3, 5

[15] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Spectre: Defending against backdoor attacks using robust statistics. In *International Conference on Machine Learning*, pages 4129–4139. PMLR, 2021. 3, 8

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 6, 7

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 6, 7

[19] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 716–734. IEEE, 2022. 2

[20] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, pages 361–372. IEEE Press, 2014. 1

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). 5

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[23] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16463–16472, 2021. 3

[24] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *International Conference on Learning Representations*, 2021. 3, 8

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 6

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 1

[27] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021. 2, 3

[28] Xiangyu Qi, Tinghao Xie, Ruizhe Pan, Jifeng Zhu, Yong Yang, and Kai Bu. Towards practical deployment-stage

backdoor attack on deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13347–13357, 2022. 1, 3

[29] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1211–1220, 2019. 3, 6

[30] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13198–13207, 2020. 1, 2, 3, 5, 6, 7, 8

[31] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7928–7939, 2021. 1

[32] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenantfpga. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1919–1936, 2021. 6

[33] Mark Sandler, Andrew Howard, M Zhu, A Zhmoginov, and LC Chen. Mobilenetv2: The next generation of on-device computer vision networks. In *CVPR*, 2018. 6, 7

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 6

[35] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. *Black Hat*, 15: 71, 2015. 2

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6, 7

[37] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021. 6, 7

[38] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, 31, 2018. 3

[39] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689, 2016. 2

[40] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. 3, 8

[41] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops:{Cross-VM} row hammer attacks and privilege escalation. In *25th USENIX security symposium (USENIX Security 16)*, pages 19–35, 2016. 2

[42] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016. 1

[43] Fan Yao, Adnan Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networksthrough targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020. 1, 3

[44] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 28–41. IEEE, 2020. 3

[45] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *International Conference on Learning Representations*, 2020. 3

[46] Mengxin Zheng, Qian Lou, and Lei Jiang. Trojvit: Trojan insertion in vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4025–4034, 2023. 2, 3, 7

[47] Runkai Zheng, Rongjun Tang, Jianze Li, and Li Liu. Data-free backdoor removal based on channel lipschitzness. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part V*, pages 175–191. Springer, 2022. 2, 3, 8