DeepVenom: Persistent DNN Backdoors Exploiting Transient Weight Perturbations in Memories

Kunbei Cai
University of Central Florida
kunbei.cai@ucf.edu

Md Hafizul Islam Chowdhuryy
University of Central Florida
hafizul.islam@ucf.edu

Zhenkai Zhang
Clemson University
zhenkai@clemson.edu

Fan Yao
University of Central Florida
fan.yao@ucf.edu

***Abstract*—Backdoor attacks have raised significant concerns in machine learning (ML) systems. Mainstream ML backdoor attacks typically involve either poisoning the victim's training samples or pre-training poisoned models for use by victim users. Meanwhile, recent advances in hardware-based threats reveal that ML model integrity at *inference-time* can be seriously tampered by inducing transient faults in model weights. However, the adversarial impacts of such hardware fault attacks at *training time* have not been well understood.**

In this paper, we present *DeepVenom*, the first end-to-end hardware-based DNN backdoor attack during victim model training. Particularly, DeepVenom can insert a targeted backdoor *persistently* at the victim model fine-tuning runtime through *transient* faults in model weight memory (via rowhammer). DeepVenom manifests in two main steps: i) an offline step that identifies weight perturbation *transferable* to the victim model using an *ensemble-based* local model bit search algorithm, and ii) an online stage that integrates advanced system-level techniques to efficiently massage weight tensors for precise rowhammer-based bit flips. DeepVenom further employs a novel *iterative backdoor boosting* mechanism that performs multiple rounds of weight perturbations to stabilize the backdoor. We implement an end-to-end DeepVenom attack in real systems with DDR3/DDR4 memories, and evaluate it using state-of-the-art Convolutional Neural Network and Vision Transformer models. The results show that DeepVenom can effectively generate backdoors in victim's fine-tuned models with upto 99.8% attack success rate (97.8% on average) using as few as 11 total weight bit flips (maximum 49). The evaluation further demonstrates that DeepVenom is successful under varying victim fine-tuning hyperparameter settings, and is highly robust against catastrophic forgetting. Our work highlights the practicality of training-time backdoors through hardware-based weight perturbation, which represents a new dimension in adversarial machine learning.

1. Introduction

Machine Learning (ML) is rapidly transforming our daily life [1]. The tremendous advances of deep neural networks (DNN) have enabled its adoption a wide range of application domains, including image processing, autonomous driving and medical diagnostics [2], [3]. As many of these techniques are applied in critical systems that aid human decision-making, ensuring the security and trustworthiness of ML-integrated computing systems is critical. Recent studies reveal the integrity of ML models can be tampered either externally (e.g., through input perturbations [4], [5], [6]) or internally (e.g., via weight perturbations [7], [8], [9], [10], [11]), leading to compromised ML model behavior during inference.

Model backdoors represent one of the most concerning classes of integrity tampering attacks in DNNs [8], [12], [13]. Trojaned ML models predict normal inputs correctly while perform maliciously (e.g., with attacker-desired classification) under inputs perturbed with certain trigger patterns, which makes such attacks extremely stealthy and challenging to eradicate. State-of-the-art DNN backdoor primarily falls into the following categories: i) data poisoning attacks [14], [15] where poisoned samples are fed into victim's training process for trojaning; ii) retraining-based trojan attacks in which the adversary locally trains and publishes a model with backdoor, which is later adopted by a victim [8], [16]. These attacks either require *direct tampering* of the victim's training samples or depend on model users to obtain well-trained/pre-trained models from *unverified sources*. Hence, they are ineffective when both data and models have been verified in advance.

Prior studies have revealed that commercial-off-the-shelf hardware (e.g., logic and memory components) is vulnerable to fault attacks [17], [18]. This threat disturbingly permits *internal tampering* of ML models at runtime, jeopardizing the security of ML systems even when dataset and model sources are trusted. For instance, recent works demonstrate that through inducing bit flips in model weight parameters (i.e., using rowhammer), attackers can manage to manipulate the inference of targeted DNN models with backdoor insertions [10], [11]. However, existing DNN fault attacks primarily manifest at *inference time*. One main limitation of inference-time attacks is that the backdoor functionality is only temporary since those hardware-induced faults are inherently *transient*. In essence, such exploitation is not permanently implanted into the victim's model. For instance, reloading the static weights from the backing storage can annul the weight perturbations in memories [19]. One critical question to raise in our community is: *will it be possible to trojan ML models at **training time** through hardware transient faults?* Specifically, we consider the novel attack scenario (shown in Figure 1) where the adversary targets vic-

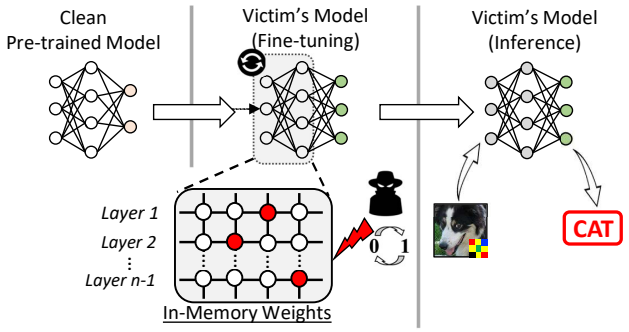


Figure 1: DeepVenom attack scenario.

tim's training runtime that involves fine-tuning a publicly-obtained pre-trained model (PTM). Note that fine-tuning from PTMs has become a widely popular approach nowadays for fast ML service deployment [20], [21], [22]. The attacker perturbs a small amount of victim model weights using transient faults in memory *during* its fine-tuning process, which aims to eventually insert a backdoor in victim's fine-tuned model. Different from inference-time exploits, training-time fault attacks have the potential promise of transforming the effect of *transient weight perturbations* to *persistent backdoors* if the weight learning/updating process is properly exploited.

Although training-time DNN backdoor is appealing from the adversary's perspective, our initial investigation unveils several fundamental challenges for such attacks. **Firstly**, the training process essentially renders fault injections in ML models at this stage a *blackbox* exploit due to the unknown and periodic changing of model weights. Therefore, the gradient-based methods (i.e., using label loss) under the whitebox setting for fault localization used in prior inference-time DNN attacks (e.g., [11], [23]) are not applicable. **Secondly**, with the unawareness of the victim's model internals during training, one might consider the possibility of generating a set of bit flips in a substitute model (whitebox) and applying them to the victim. However, while a large body of prior works has demonstrated successful transfer of input perturbations across models (i.e., adversarial input [4], [5], [6], [13]), no existing works have investigated the *transferability of weight perturbations*. It is an open problem whether perturbations of weights can transfer at all. **Last but not least**, it is well known that ML training exhibits the phenomenon of *catastrophic forgetting* [24] where certain learned information (via perturbed weights in this case) can be forgotten as new information is learned (e.g., the regular training task).

In this paper, we answer the aforementioned question and overcome the associated challenges by presenting the first end-to-end hardware fault-based DNN backdoor attack during training time–DeepVenom. At a high level, DeepVenom is a multi-round attack that identifies bit flips in model weights during local training (offline stage) and subsequently applies the fault injections to the bits (i.e., transfers the weight perturbation) of the victim's model during fine-tuning (online stage). Specifically, **in the offline stage**, the attacker constructs a local model that resembles the fine-tuning process of the victim. DeepVenom generates a transferable input trigger and bit flip group based on the local model. Targeting a snapshot of the local model at certain time, the proposed algorithm aligns signature neurons in the *intermediate representation* with those of the clean inputs from the targeted class, using weight bit flips. To further enhance the transferability of the generated trigger and bit flip group, DeepVenom integrates a novel technique utilizing multiple local models in an *ensemble fashion* that 1) generates advanced input trigger that can activate individual set of signature neurons on different instances of the fine-tuned models and 2) leverages a fused bit search loss among local models to locate *highly invariant bits* in model parameters for enhanced weight perturbation transferability. Lastly, to mitigate catastrophic forgetting, DeepVenom utilizes an *iterative backdoor boosting* technique that identifies and applies multiple bit flip groups at different time points for backdoor stabilization. In the **online stage**, DeepVenom integrates novel rowhammer-based techniques that can efficiently massage ML model weight tensors for rowhammer bit flips in real systems.

We evaluate our attack on four representative DNN model architectures (VGG16, ResNet18/50, and ViT) and five distinct datasets (GTSRB, CIFAR10, SVHN, EuroSat, and CIFAR100). The experimental results demonstrate that DeepVenom can successfully insert backdoors into the victim model with a high attack success rate (97.8% on average) and negligible accuracy drops (0.1% on average) for normal inputs. DeepVenom can achieve stable backdoors using rowhammer, requiring as few as **11 flips** (maximum **49**). Moreover, our evaluation shows that the DeepVenom backdoor persists in the victim model even when local and victim fine-tuning hyper-parameters differ (e.g., optimizer and learning rate). Additionally, our results indicate that *iterative backdoor boosting* is highly effective in defending the attack against catastrophic forgetting: the inserted backdoor remains stable after further extended fine-tuning by the victim. The following are our main contributions:

- We perform the first study to investigate the feasibility of trojaning DNN models through hardware-based fault attacks in *training stage* under a transfer learning scenario and identify unique challenges for such attacks.
- We explore factors that impact the transferability of weight perturbations and propose a novel algorithm that generates input trigger and bit flips based on local training, and further optimizes their transferability to the victim using an ensemble approach. By taking the advantage of *runtime* exploitation (with rowhammer), the algorithm performs *iterative backdoor boosting* that carries out multi-round fault injections to induce a highly stable backdoor.
- We build a prototype of DeepVenom and implement the end-to-end attack in real systems with both DDR3 and DDR4 memories. By identifying the memory management mechanisms in state-of-the-art ML platforms (i.e., PyTorch), DeepVenom employs a novel memory

massaging technique tailored to ML training to induce deterministic bit flips in global weight tensors. The results show DeepVenom is highly successful in trojaning the victim model with high ASRs (up to 99.8%) and only **11 to 49** bit flips throughout victim's fine-tuning.
- Our analysis shows that the iterative backdoor boosting is particularly effective in retaining the backdoor function even when extended fine-tuning is performed.
- Finally, we discuss the applicability of existing backdoor defenses to DeepVenom and also present a potential mitigation and its effectiveness. Our work highlights the importance of understanding model integrity threat with respect to hardware vulnerabilities in the machine learning training stage.

2. Background and Related Works

2.1. Pre-trained Models and Fine Tuning

Training production-level ML models (especially for DNNs) typically not only requires substantial computing resources but may also need access to massive training dataset, which can be prohibitively expensive [25], [26], [27]. Hence, it becomes a common practice for users to leverage *pre-trained models* (PTM) downloaded from public sources (e.g., ModelZoo and HuggingFace [28], [29]). Generally, there are two PTM use cases: *scenario 1*, deploying the publicly-obtained PTMs directly for an ML service (i.e., inference) [8], [16], and *scenario 2*, fine-tune the model (i.e., transfer learning) for certain downstream task [30], [31], [32], [33], [34]. Due to the common need for ML service customization, fine-tuning PTM has become a widely popular approach for service providers [20]. Specifically, in the fine-tuning process, the last layer(s) of the model are first replaced with new ones tailored to the output classes of the downstream task. Depending on the difference between the upstream/downstream tasks and the size of downstream task dataset, updates of weights can be performed either in a limited fashion (e.g., only modifying weights in the last a few layers and keeping the rest unchanged) or onto all weights during fine-tuning. It has been shown that fine-tuning all layers can help improve the generality and render better performance in transfer learning [35]. Figure 2 illustrates the high-level procedure of fine-tuning from pre-trained models.

2.2. Backdoor Attacks

Backdoor attacks aim to insert stealthy trojan into an ML model such that it infers normally on clean inputs but generates unexpected outputs when certain pre-determined trigger is applied to an input [8]. Attackers can manipulate the infected model to behave maliciously in different ways including: 1) *untargeted backdoors* that induce misclassification to an arbitrary class given a triggered input [33], [34], [36], and 2) *targeted backdoors* where any input with the trigger is predicted to an attacker-desired class [8], [14], [16]. Traditionally, DNN backdoor attacks are performed

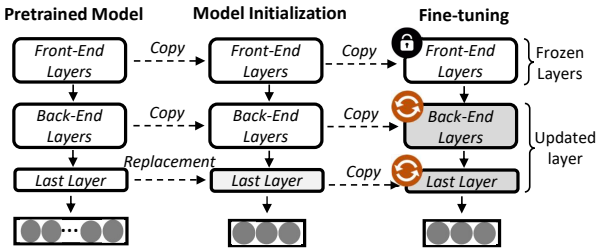


Figure 2: Transfer learning with pre-trained models.

through either tampering the training data [8], [14] or the training process [16], [32]. In data poisoning attacks, the victim's training dataset is maliciously polluted by samples with embedded trigger patterns [14], [37]. Moreover, attackers can backdoor a model by controlling/compromising the training procedure. In particular, attackers can publish a locally-trained model (i.e., malicious PTM) to be used by the victim users [7], [8], [16]. Recent works also demonstrated the possibility of hijacking the training function at runtime for backdoor insertion [38]).

With the growing demand of utilizing PTMs for building downstream tasks, recent ML security studies have investigated DNN backdoors in the context of transfer learning [30], [31], [32], [33], [34], [36], [39]. In such scenarios, the attacker maliciously constructs (via local training) a PTM with a backdoor not aimed to target the model itself, but *to manifest in the fine-tuned model* used by a victim. For instance, prior works [30], [31] have shown that targeted backdoors can be realized by trojaning a pre-trained model (through re-training) to force feature map similarity between triggered inputs and the clean inputs for target class. However, these attacks are successful only when fine-tuning is limited to the last layer(s) [30] (See Section 2.1). Achieving backdoors after fine-tuning of all weights is very challenging [37], [40], [41], [42], [43]. This is because the model is put to *only* learn the new feature corresponding to the downstream tasks since only the downstream dataset is fed into the training pipeline. As such, the backdoor functionality inserted in the PTM at its initial state will be gradually weakened due to the widely-known effect of *catastrophic forgetting* [24], [44], [45], [46]. Several recent studies explore ways to mitigate the forgetting effect by modeling and alleviating the weight update conflicts between backdoor and downstream task [32], [39]. While these approaches demonstrate backdoor stability enhancement in certain settings, the effectiveness of these attacks is significantly limited when there is a diversion between the local and victim model configurations (e.g., learning rates).

It is worth noting that the aforementioned backdoor attacks either rely on victim users to access impacted dataset/pre-trained models or depend on the training routine to be hijacked by adversaries. Hence, such attacks cannot manifest in use cases where users obtain PTMs from trusted sources and utilize audited or untampered data [38].

2.3. Hardware-based Attacks on ML Models

Hardware-based Faults Attacks. It has been known that computing hardware is vulnerable to fault injections. Hardware faults can be injected into a variety of components, including on-chip logic [47], [48], SRAMs [49] and off-chip memory [18]. Particularly, recent advances have shown that commercial-off-the-shelf DRAM devices, which are the building block for system main memory, are vulnerable to bit flip-based fault injection (known as rowhammer [18], [19], [50], [51], [52]). Specifically, it has been observed that activations of one DRAM row accelerates the charge leakage of DRAM cells in the physically-adjacent rows [18]. If such disturbance is frequent enough, it can introduce sufficient loss of charge in neighboring cells, leading to bit flips. Rowhammer is extremely worrisome as it allows *deterministic* fault injection to program memories, which can be performed *remotely* with an attacker-controlled *unprivileged process*. Recent developments have shown successful exploitation of rowhammer in a variety of DRAM technologies, including DDR3 [18], [19], [50], [51] and DDR4 [19], [52] memories. More alarmingly, it has been revealed that with further scaling of technology nodes, future DRAMs are expected to become increasingly susceptible to rowhammer [53]. While various studies have proposed techniques to mitigate such a vulnerability [54], [55], [56], [57], they either do not fully eliminate bit flips or can incur non-trivial hardware/runtime cost.

ML Model Tampering with Bit Flips. The capability of precise and fine-grained fault injection with rowhammer opens a new dimension of *adversarial machine learning*. Essentially, rowhammer enables practical *internal tampering* of DNN models through *weight perturbations* in contrast to *input perturbations* (i.e., adversarial examples [4], [13], [58]). In fact, recent studies have shown that by inducing only a few bits among millions of weight parameters, attackers can drastically degrade the inference accuracy in state-of-the-art DNN models [10], [19], [59], [60], [61], [62]. More recent advances have revealed the possibility of embedding backdoors by flipping bits in the static weights of models [11], [23], [61], [63], [64], [65]. For example, the attack in [23] identifies sensitive neurons to target class and locates bit flips in the corresponding weights of the last layer to divert the prediction of inputs with trigger to the desired label. Tol et al. [63] proposes to co-optimize trigger and bit flips to directly associate the triggered inputs with the target class. While these works showcase the practicality of model tampering through bit flips in memory, we note that they have several key limitations. Firstly, since rowhammer only causes soft errors in DRAM devices [18], the induced fault injections are *inherently transient*. Therefore, the inserted backdoor is considered only temporary and cannot persist across model weight reloads or model migration from one machine to another. Secondly, such attacks typically assume white-box settings where weights are *static* and precisely known to the attacker, limiting its applicability to model training where weights are dynamic and the exact values are uncertain at runtime.

3. Threat Model

In this paper, we investigate a *new attack direction*: backdoor a victim model *during* its training/fine-tuning through transient weight perturbation via rowhammer. Our attack targets a common use case where a victim user adopts transfer learning and fine-tunes a pre-trained model *free of backdoors* (e.g., from trusted source) using *clean* or *sanitized* downstream dataset. We assume that the pre-trained model is known by the attacker. Note that such assumption is reasonable due to the popularity of using publicly-accessible PTMs to build ML service [20], [21]. Without loss of generality, our main discussion assumes that for fine-tuning, the last layer(s) of the PTM is replaced with a new layer for the downstream task. Note that our attack can also be applicable in case multiple layers are used for the new classifier. We assume the victim fine-tunes *all layers* of the PTM with weights in *floating points*, which is the most general fine-tuning strategy. The adversary manages to co-reside on the same machine as the victim user (e.g., in the cloud). Moreover, the attack process is unprivileged and has no control over the victim's fine-tuning operations and hyper-parameters (e.g., training duration and learning rate). The adversary intends to eventually leave a *persistent and targeted backdoor* in the victim's fine-tuned downstream model. To demonstrate an end-to-end attack, we mainly target victim fine-tuning on CPUs where model weights are stored in system main memory.

We assume the attacker is aware of the victim's downstream task and has access to a very small publicly available portion of the victim's dataset. Note that our assumption of access to limited dataset is less stringent compared to prior re-training based targeted backdoors that assumes full dataset access [32], [39]. Furthermore, to inject faults into weights at runtime, we assume the attacker can monitor and estimate victim's fine-tuning progress at a coarse-granularity (e.g., detecting the start of a fine-tuning iteration), which has been shown to be plausible using microarchitectural side channels (e.g., cache attacks) in recent studies [66], [67], [68]. Such information is used in the attack process to induce weight bit flips at certain times of the fine-tuning. We note that a precise tracking of the ongoing epochs/iterations is **not** a requirement for our attack.

4. DeepVenom Overview

In this section, we present an overview of DeepVenom that aims to embed backdoors by injecting bit flips to weights at the runtime of fine-tuning, which can be later activated at the inference time of the fine-tuned model.

Attack Overview. Figure 3 shows the overview of our proposed attack. At a high level, DeepVenom manifests as a *multi-round* attack framework with an **offline stage** (for trigger and bit flips generation) and an **online stage** (for rowhammer-based fault injection). The offline stage involves the attacker's local fine-tuning of the public PTM (to be used by the victim). At certain time of the local fine-tuning,

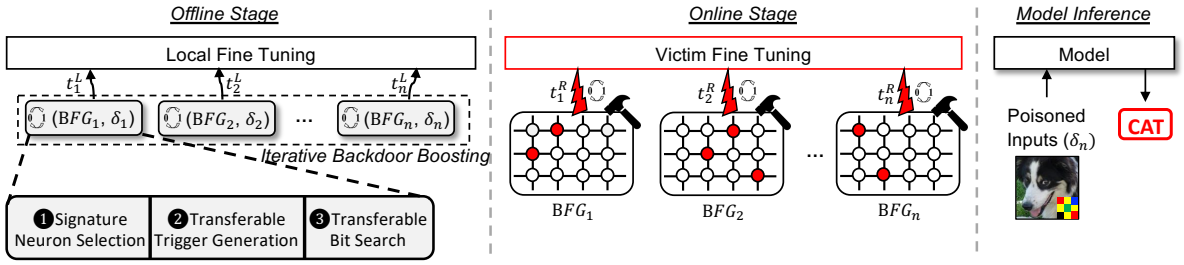


Figure 3: Overview of the proposed DeepVenom attack framework.

DeepVenom identifies a group of weight bits to flip, called a *bit flip group* (BFG) together with the generation of an input trigger (δ) that are independent of the model's last layer(s) (Section 5.1). BFG and δ are further optimized through an ensemble method, enhancing both weight and input perturbation transferability for the backdoor (Section 5.2). Weight bits in BFG are inverted in the snapshot local model (fully controlled by the attacker), and the immediate backdoor effectiveness is assessed using δ. To tackle catastrophic forgetting, DeepVenom adopts *iterative backdoor boosting* that accumulates multiple rounds of the above attack at different fine-tuning times until the backdoor feature in the local model is stabilized (Section 5.3). Specifically, DeepVenom identifies (δ_i, BFG_i) at local fine-tuning time t_i^L for attack round i ($i \in [1, n]$). In the online stage, DeepVenom leverages rowhammer to correspondingly perform the multi-round attack during the victim's model fine-tuning, each flipping bits in one BFG (i.e., BFG_i for round i) at the remote victim's fine-tuning time t_i^R. This eventually inserts a persistent backdoor in the victim's model (Section 6).

5. DeepVenom Offline Stage

This section describes the offline attack procedure that generates transferable input trigger and bit flip groups in order to backdoor victim's fine-tuning at runtime.

5.1. Attacking a Local Fine-tuning Snapshot

In the offline stage, the attacker first downloads the public clean PTM $\boldsymbol{M}$ with l layers. To prepare for local fine-tuning, the attacker replaces the last layer of $\boldsymbol{M}$ with a new one (L_a) based on the victim's task to generate a new initial model $\{\boldsymbol{M}^a\}_0$, with which the attacker starts the local fine-tuning. At certain fine-tuning time t with the snapshot of attacker's local model $\{\boldsymbol{M}^a\}_t$, the attacker manages to derive a trigger pattern (δ) and BFG such that $\{\boldsymbol{M}^a\}_t$ with flipped bits in BFG will embed a backdoor to be activated by inputs with trigger (δ). Note that at the online stage, the victim will replace the last layer of $\boldsymbol{M}$ with a *randomly initialized* task-specific layer L_v (i.e., weights unknown to the attacker) to configure her own model $\{\boldsymbol{M}^v\}_0$ for fine-tuning. The key objective is to ensure that the backdoor in the local snapshot model ($\{\boldsymbol{M}^a\}_t$) can: ❶ transfer to the victim's model when BFG is applied to victim's weights *amid fine-tuning*, and ❷ be able to persist to the end (i.e, the final fine-tuned model ($\{\boldsymbol{M}^v\}_{end}$). We discuss the setting of attack times in local and victim fine-tuning in Section 5.3.

Existing bit flip-based backdoors manifest at *inference time* and are highly dependent on *weights of the last layer(s)* (i.e., classifier) [10], [11], [23], [63], [64]. This is because they either rely on directly flipping bits in the classifier [23] or involve weights of the classifier in the *backdoor loss* computation [11]. While such algorithms are effective for while-box DNN models, the nature of model training determines that the exact weight values in the victim's classifier layer (and hence the whole model) during fine-tuning are not known to the attacker, albeit $\{\boldsymbol{M}^a_{1:l-1}\}_0$ and $\{\boldsymbol{M}^v_{1:l-1}\}_0$ being the same. Therefore, triggers and bit flips derived in prior works would not effectively transfer the backdoor to the victim's model in our attack scenario (See Appendix A for detailed analysis). In this work, we propose an algorithm that generates *transferable* weight perturbation via bit flips.

5.1.1. Signature Neuron Selection (SNS). Without knowing the classifier layer, an intuitive approach is to connect the triggered input to the *output feature* (Φ) of $\{\boldsymbol{M}^a_{l-1}\}_t$ (i.e., intermediate representation *right before* the classifier) for the target class ξ, similar to the mechanisms in prior re-training based PTM backdoor [30], [31], [39] (Section 2.2). However, given the limited impact of weight perturbation at bit level, it is infeasible to control the entire feature map via flipping a reasonable amount of weight bits. To tackle this challenge, our algorithm selects a few *signature neurons* in Φ that best represent the feature map for a specific class. The main idea is to enforce high similarity at the **signature neuron-level** between triggered inputs and clean inputs for class ξ in the local model. Since the local and victim model are under fine-tuning from the same initial state $\boldsymbol{M}_{1:l-1}$, the feature of an input in the local model is highly likely to resemble that of the victim's model. Hence, through activating the similar signature neurons in the victim model, triggered inputs will be classified as target class ξ by the subsequent classifier, independent of its weight values. Such mechanism has the promise of transfer the backdoor in transfer learning.

Generally, neurons with large values are more representative than smaller ones [16]. Accordingly, SNS aims to identify neuron locations with relatively-large values for the target class ξ. In particular, we compute $\Phi(\boldsymbol{x})$ for each input $\boldsymbol{x}$ in dataset $\mathbb{D}$. The average feature map Φ^ξ belonging to inputs from class ξ is calculated by averaging all the feature

maps. Using the same method, the average feature map $\Phi^{\bar{\xi}}$ is then generated for the inputs in classes other than ξ. To select the neuron locations with relatively large value, the attacker computes the element-wise difference of Φ^{ξ} and $\Phi^{\bar{\xi}}$ to derive the relative feature map Φ^{r}. Finally, p neuron locations with the top neuron values are identified in Φ^{r}, denoted as Γ. Essentially, Γ points to the *signature neurons* for clean inputs belonging in target class ξ.

5.1.2. Transferable Trigger Generation (TTG). With the identified signature neurons, this step generates a trigger optimized for enhancing the *association between triggered inputs and the signature neurons*. We consider a trigger pattern δ as a patch stamped onto a corner of input samples. Specifically, we define a binary matrix mask $\boldsymbol{\tau}$ to denote the shape and size of the pre-determined trigger. An element-wise multiplication $\boldsymbol{x} \circ (1 - \boldsymbol{\tau})$ is conducted to delineate the region of the clean sample. The triggered input can be defined as:

$$\begin{aligned} \boldsymbol{x}^{*} &= \Psi(\boldsymbol{x}, \boldsymbol{\tau}, \delta) \\ \Psi(\boldsymbol{x}, \boldsymbol{\tau}, \delta) &= \boldsymbol{x} \circ (1 - \boldsymbol{\tau}) + \delta \circ \boldsymbol{\tau} \end{aligned} \tag{1}$$

where function $\Psi()$ is used to synthesize inputs with trigger, $\boldsymbol{x}$ is the clean input. The trigger generation can be formally expressed as an optimization problem:

$$\begin{aligned} \ell_{\delta} &= \mathcal{L}\left(f(\boldsymbol{x}^{*}; \{\boldsymbol{W}\}_{i=1}^{l-1})_{\Gamma}, c\right) \\ \delta &= \arg\min_{\delta}(\ell_{\delta}) \end{aligned} \tag{2}$$

where $\{\boldsymbol{W}\}_{i=1}^{l-1}$ is the snapshot weights of substitute model $\{\boldsymbol{M}^{a}\}_{t}$ from 1 to $l-1$ layers. For a given triggered input $\boldsymbol{x}^{*}$, we first calculate the feature map of the sample via forward propagation. $f(\boldsymbol{x}^{*}; \{\boldsymbol{W}\}_{i=1}^{l-1})_{\Gamma}$ denotes neuron values at locations in Γ, and c is the maximum value of the signature neurons in Φ_{Γ}^{ξ}. We use the Projected Gradient Descent algorithm [58] to derive the trigger to optimize ℓ_{δ}.

5.1.3. Transferable Bit Flip Identification (TBFI). Once the trigger is obtained, the next step is to generate the BFG from the local model ($\{\boldsymbol{M}^{a}\}_{t}$). DeepVenom aims to perform the minimum number of bit flips for transferring the backdoor. To do so, TBFI finds one vulnerable weight bit at a time with the following bit flip loss function:

$$\begin{aligned} \ell_{B} &= \underbrace{\mathcal{L}\left(f(\boldsymbol{x}^{*}; \{\boldsymbol{B}\}_{i=1}^{l-1})_{\Gamma}, c\right)}_{\text{backdoor loss: } \ell_p} + \lambda \cdot \underbrace{\mathcal{L}\left(f(\boldsymbol{x}; \{\boldsymbol{B}\}), y\right)}_{\text{downstream task loss: } \ell_c} \\ \hat{\boldsymbol{B}} &= \arg\min_{\boldsymbol{B}} \ell_{B}; \quad Dist(\hat{\boldsymbol{B}}, \boldsymbol{B}) = 1 \end{aligned} \tag{3}$$

where the ℓ_p term is the poisoning loss for the backdoor task, and ℓ_c denotes the clean loss for downstream task. y is the ground truth label for benign input $\boldsymbol{x}$. Moreover, $\boldsymbol{B}$ is the binary weight representation of $\{\boldsymbol{M}^{a}\}_{t}$, and $\hat{\boldsymbol{B}}$ is the model with one weight bit flipped. $Dist()$ measures the number of different bits between the two models. Essentially, with the bit flip, ℓ_p further optimizes the neuron values of $\boldsymbol{x}^{*}$ at location Γ while ℓ_c maintains the normal accuracy.

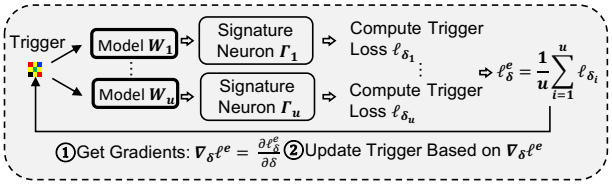
Identifying BFG. TBFI searches one weight bit iteratively in the local model. Specifically, the algorithm first computes ℓ_B given trigger δ and dataset $\mathbb{D}$, then back-propagates the loss to get the *weight-level* gradients $\nabla_{\boldsymbol{B}}\ell$. Since weight perturbation occurs at bit level with rowhammer, an additional mechanism is needed to rank the contribution to ℓ_B if a weight bit is flipped. For a weight w with gradient ∇w and weight change Δw (caused by flipping certain bit in w), we quantify the influence of the flip using the following score: $\Delta w \cdot \nabla w$. TBFI considers weight bit candidates with the top q highest scores in each layer, which results in $\boldsymbol{q} \times \boldsymbol{l}$ total candidate bits. TBFI computes the loss value ℓ_B after flipping each candidate bit individually. The algorithm then selects the most influential bit that induces the minimum loss if flipped. At the end of this iteration, the identified bit flip f is applied in the local model (via direct modification) and appended to BFG. f includes both the *global offset of the weight bit* as well as the flip direction (e.g., '0'→ '1' or '1'→ '0'). TBFI performs this search for multiple iterations until the local attack success rate (ASR) is higher than a threshold asr^{T}, forming a bit flip group BFG : $\{f_i \mid i = 1, 2, \ldots, k\}$.

Selecting Highly-invariant and Suitable Candidate Bit Offsets. Compared to prior transferability studies on input perturbation in adversarial examples [4], [5], [6], [13], transfer of weight perturbation from local to victim model is more challenging. This is because successful transfer of weight perturbations needs to satisfy two conditions: ❶ the weight bits in BFG should be *flippable* in the victim model; ❷ the effect of bit flips (i.e., the backdoor feature) should also transfer. We create two metrics: *flipping success rate* ($\mathcal{R}$) and *effect transferability* ($\mathcal{E}$) for weight perturbation transferability, defined as the following:

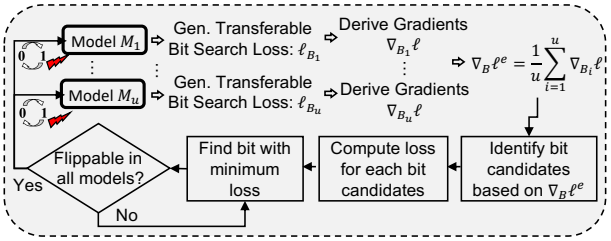
$$\begin{aligned} \mathcal{R} &= \frac{1}{k} Dist(\boldsymbol{B}_{v}^{k}, \boldsymbol{B}_{v}) \\ \mathcal{E} &= \mathbb{P}[f(\boldsymbol{x}^{*}; \boldsymbol{B}_{v}^{k}) = \xi] \end{aligned} \tag{4}$$

where $\boldsymbol{B}_v$ is the binary representation of the snapshot weights of victim's model $\{\boldsymbol{M}^{v}\}_{t'}$ (assuming runtime attack time at t'), and $\boldsymbol{B}_{v}^{k}$ represents the binary weights with k *bit flips attempted* according to the BFG. $\mathbb{P}$ is the percentage of triggered inputs $\boldsymbol{x}^{*}$ being classified to the target class after applying the weight perturbations to $\boldsymbol{B}$. Note that victim's model parameters are periodically updated during fine-tuning and will certainly differ from local model $\{\boldsymbol{M}^{a}\}_{t}$. Hence, a flip f_i may not be successfully applied to $\boldsymbol{B}_v$. Particularly, consider a flip f_i that specifies to flip a weight bit b_i with a binary value '1' at bit offset o with the intended flip direction ('1'→'0'). However, at victim's runtime time t', if b_i is updated to '0' by victim's regular fine-tuning, the flipping would not be successful and the backdoor effect would not transfer (i.e., failure of $\mathcal{R}$ transferability).

To address the transferability issue, DeepVenom needs to identify weight bits that remain highly-invariant throughout the fine-tuning process. That is, b_i ($i \in [1, k]$) should remain constant across different fine-tuning instances starting from the same PTM. It is worth noting that training/fine-tuning is



(a) Ensemble Trigger Generation



(b) Ensemble BFG Identification

Figure 4: Ensemble trigger and BFG identification.

typically performed on floating point numbers. Fortunately, our characterization study on weight changes indicate that while PTM fine-tuning updates all weight parameters, the magnitude of changes are very small. As a result, bits at certain offsets of the *exponent segment* in weight parameters are largely unaltered after fine-tuning. Note that not all exponent bits are suitable for flipping due to the considerable dynamic value range for floating points. We formulate FP32 weight change due to a flip in certain offset in the following:

$$\Delta w = (S_o^d - 1) \cdot w \tag{5}$$

where w is the original weight value. S_o^d denotes the scaling factor for w when flipping occurs at offset o in an FP32 parameter with flip direction d (**0**: '0'→'1' or **1**: '1'→'0'). Note that for fixed o and d, S_o^d is a unique value regardless of w. We observe that flipping high order exponent weight bit from '0'→'1' introduces significant weight change (e.g., $S_{30}^0 = 2^{128}$), which can completely malfunction the model (e.g., failure of regular training). Our investigation shows that considering bit invariance and reasonable weight change, the 1^{st}, 2^{nd} and 3^{rd} least significant bits of weight exponents are suitable candidates for locating transferable flips (See Appendix B for more discussion). Therefore, TBFI empirically limits the bit search and ranking within the weight bits at these offsets.

5.2. Enhancing Transferability with Ensemble Models

The basic DeepVenom in Section 5.1 leverages a single substitute model for transferable trigger and BFG identification. However, relying solely on one local model can be suboptimal for weight perturbation transferability. Specifically, we observe that during regular fine-tuning, feature map Φ^ξ sometimes exhibits *non-trivial and uncertain pattern shift* among distinctive downstream model instances, leading to difference in signature neuron locations. Such discrepancies can undermine the transferability of weight perturbation particularly for $\mathcal{E}$ (Section 5.1). Recent studies on adversarial examples [69], [70], [71] indicate transferability of input perturbation can be enhanced with the use of multiple local models. Motivated by such observation, we further improve DeepVenom with a novel design of ensemble models tailored for improving trigger and BFG transferability.

Specifically, DeepVenom constructs u substitute models that start fine-tuning in parallel and progress in synchronization. Each substitute model is independently initiated with a random replaced last layer. At the local time t, u snapshots of substitute models are obtained. Under the ensemble mode, one way to adapt *Signature Neuron Selection* is to utilize shared *Signature Neurons* among all u models. However, such approach may negatively affect the representativeness of the selected neurons due to variations among Γ_i for $i \in [1 : u]$. Therefore, we choose to maintain distinct signature neurons Γ for each local model. To generate the ensemble trigger, DeepVenom leverages the *ensemble trigger loss* defined in the following:

$$\delta^e = \arg\min_{\delta} \left(\frac{1}{u} \sum_{i=1}^{u} \ell_{\delta_i}(f(\boldsymbol{x}^*; \{\boldsymbol{W}_i\}_{j=1}^{l-1})_{\Gamma_i}, c_i) \right) \tag{6}$$

Figure 4a illustrates the ensemble trigger generation procedure. Specifically, the algorithm computes trigger loss ℓ_{δ_i} using a per-model c_i for each model $\boldsymbol{W}_i$. It then gets the *ensemble* trigger loss ℓ_δ^e by averaging among all ℓ_{δ_i}, which is used to derive the corresponding ensemble gradient $\nabla_\delta \ell^e$. Finally, the **ensemble trigger** δ^e is computed based on $\nabla_\delta \ell^e$. The derived δ^e is expected to activate *individual set* of signature neurons on different local model instances for the target class. For ensemble bit search, the algorithm calculates the ensemble bit flip loss ℓ_B^e with the the average of ℓ_{B_i} for $i \in [1 : u]$:

$$\ell_B^e = \frac{1}{u} \sum_{i=1}^{u} \ell_{B_i}(\boldsymbol{x}, \boldsymbol{x}^*, \boldsymbol{B}_i, c_i, y) \tag{7}$$

The fused gradient $\nabla_{\boldsymbol{B}} \ell^e$ (w.r.t. weights) is then generated using ℓ_B^e. Afterwards, the top q weight candidates per layer are identified by ranking $\Delta w \cdot \nabla w^e$ at each layer where ∇w^e is retrieved from $\nabla_{\boldsymbol{B}} \ell^e$. A bit that can be flipped *among all local models* and introduce the maximum ensemble bit flip loss drop is selected. The flip is then applied on the corresponding weight parameter across models (with weight values not necessarily the same). An individual ASR (asr_i) is evaluated under inputs with trigger δ^e for each local model to get the *average ASR* (asr^e). The same iterative search process as in the single model TBFI (Section 5.1.3) is performed until asr^e exceeds the threshold asr^T. Figure 4b illustrates the process of ensemble bit search process. Finally, Algorithm 1 describes the detailed steps of our ensemble method. The ensemble method selects weight bit that can reliably contribute to the backdoor function universally across multiple local instances, thus having the promise of being more robust against the uncertainty in victim's fine-tuning and improving the $\mathcal{E}$ transferability.

Algorithm 1: Snapshot attack on ensemble models

Input : Dataset $\mathbb{D}$, u local snapshot models $\{\boldsymbol{W}_1, \boldsymbol{W}_2, \dots \boldsymbol{W}_u\}$ with binary weights $\{\boldsymbol{B}_1, \boldsymbol{B}_2, \dots \boldsymbol{B}_u\}$, threshold asr^T
Output: Trigger δ^e and bit flip group BFG

```
// Signature neuron selection
```
Get signature neuron locations Γ_i and target signature neuron values c_i for each local model $\boldsymbol{W}_i$
```
// Ensemble trigger optimization
```
Get ensemble trigger δ^e based on Eq. 6
```
// Ensemble bit search
```
while $asr^e \leq asr^T$ **do**
  Get ensemble bit flip loss ℓ_B^e using Eq. 7
  Get ensemble gradients $\nabla_B \ell^e = \frac{1}{u}\sum_{i=1}^{u} \nabla_{B_i} \ell$
  Search top q candidate bits for each layer based on $\Delta w \cdot \nabla w^e$
  Compute ℓ_B^e for each candidate bit flipping
  Identify a flip f flippable to all models with the largest ℓ_B^e drop
  Apply f to u models and append f to BFG
  Evaluate ASR with δ^e and $\mathbb{D}$ on u models
  Compute the average of u ASRs as asr^e
return δ^e, BFG

Algorithm 2: Iterative backdoor boosting

Input : Dataset $\mathbb{D}$, local fine-tuning model(s), total training iterations I, test times $\{t_1, t_2, \dots, t_m\}$, threshold asr^T
Output: A sequence P with elements of $\{t^L, \delta, \mathrm{BFG}\}$

```
// Multi-round snapshot attacks
```
Generate an initial random trigger δ_0
Initialize training iteration $itr = 0$, attack round $i = 0$
```
// attacks amid local fine-tuning
```
for *current training iteration itr from 1* **to** I **do**
  if *itr in* $\{t_1, t_2, \cdots, t_m\}$ **then**
    Test ASR (asr) using δ_i and $\mathbb{D}$ on local model(s)
    if $asr < asr^T$ **then**
      `// new local attack round`
      $i \leftarrow i + 1$; $\delta_i, \mathrm{BFG_i} \leftarrow$ SnapshotAttack on current local model(s)
      $t_i^L \leftarrow itr$; Append $\{t_i^L, \delta_i, \mathrm{BFG_i}\}$ to P
  else
    Continue normal fine-tuning

Note that Algorithm 1 naturally falls back to a single model attack if only one local model is used.

5.3. Iterative Backdoor Boosting

In the transfer learning attack scenario, the adversary cannot control the victim's fine-tuning process (e.g., training end time). Although one BFG may raise the *immediate* ASR of the victim model at a high level, the victim's subsequent fine-tuning iterations can gradually dilute the backdoor effectiveness. Essentially, the weight changes due to bit flips can be considered as *a short burst of training* for the backdoor task, which is subject to the common phenomenon of catastrophic forgetting [24]. Fortunately, DeepVenom, being a runtime attack, offers a unique advantage over existing training-based backdoor attacks [8], [16], [32], [33], [34], [36], [39]. That is, DeepVenom can manifest multiple rounds of weight perturbation during victim runtime in order to enhance the backdoor feature.

DeepVenom integrates an *Iterative Backdoor Boosting* (IBB) mechanism that accumulates multiple BFGs generated from snapshot attacks launched at different fine-tuning times. Notably, in DeepVenom, the *current training iteration* is used to represent time during fine-tuning. In the offline stage, the attack initializes at a preset time t_1 (i.e., local fine-tuning has passed $t_1 - 1$ iterations). Starting from t_1, DeepVenom evaluates the local attack after every v fine-tuning iterations to check if the local ASR (or asr^e with ensemble models) exceeds the target ASR (i.e., asr^T). This results in a sequence of test rounds at times $\{t_i \mid i = 1, 2, \dots, m\}$ ($t_{i+1} = t_i + v$). At each test time t_i, if the target ASR is found to be maintained, the snapshot attack is skipped at the current iteration. Otherwise, a new *round of attack* at time t_i is mounted on the current model snapshot(s), generating an updated δ and a new BFG. This procedure is continued till the end of local training. With IBB, DeepVenom yields an n-round attack with a **perturbation schedule**: $\{t_i^L, \delta_i, \mathrm{BFG}_i\}$ for i from 1 to n where t_i^L is the time for the i^{th} attack round. Notably, the first attack round typically takes place at the first test round (i.e., $t_1^L = t_1$). Algorithm 2 illustrates the main steps of IBB.

6. DeepVenom Online Stage: Persisting Backdoors with Rowhammer

In the online stage, the attacker performs the multi-round weight perturbation using the *perturbation schedule* derived from the offline stage. Since the input trigger is only used at inference time of the victim's fine-tuned model, the online attack essentially only needs $\{t_i^L, \mathrm{BFG}_i\}$ ($i \in [1, n]$). A key consideration is to determine the online fault attack time t_i^R for each attack round i. By default, DeepVenom directly maps the local attack times to those of the victim (i.e., $t_i^R = t_i^L$, $i \in [1, n]$). This synchronized-time attack requires the attacker to track victim's fine-tuning progress from the beginning, which can be achieved with a microarchitectural side channel. Moreover, in our evaluation we observe that such attack time synchronization can be significantly relaxed without incurring noticeable impact on the attack effectiveness (See Section 8.2.3 for details). Applying a BFG at runtime involves inducing bit-level faults in victim model's weights stored the memory during fine-tuning. Although rowhammer is a well-understood attack vector for memory fault injection [18], [50], [72], mounting it against modern ML training routines poses several non-trivial challenges that have not been studied. This includes: (i) identifying the exact in-memory locations of the victim weights, i.e., the victim weight tensors and their corresponding page offsets for targeted weight bits; and (ii) accurately relocating the weights to flippable locations across all model layers. We

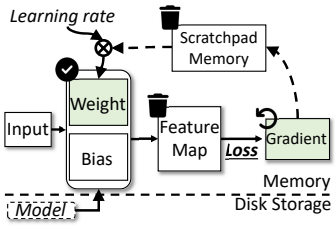


Figure 5: Memory management during training in PyTorch. - Memory objects are deallocated after every iteration; - Memory objects are reset after every iteration; - Persistent memory objects.

propose a generic three-stage procedure that is applicable to mainstream ML training frameworks as follows:

Stage 1: Memory Profiling. Similar to prior rowhammer exploitation [19], [50], [51], [52], [67], we first profile DRAMs in the victim's machine to identify cells susceptible to bit flips. This consists of two main steps: (1) occupying physically consecutive rows in DRAM to create rowhammer memory layout, and (2) setting up alternating data patterns in aggressor and target rows. Finally, the aggressor rows are hammered by rapidly accessing them alternately to observe bit flips in the target row and detect vulnerable cells. The identified vulnerable cells form a *bit flip profile*.

Stage 2: DeepVenom Memory Massaging. Once the *bit flip profile* is generated, the subsequent stage is to identify victim's weight data, specifically pages containing critical bits targeted in the BFGs, and massage them into the target row containing vulnerable DRAM cells in the same offset as the critical bit. Compared to state-of-the-art bit flip attack in ML frameworks (i.e., inference-time exploitation [19], [67]), we find that memory massaging for model training/fine-tuning presents the following unique challenges:

i) Reverse-engineering Memory Management: Prior studies have relied on the memory management policy of ML frameworks during inference (e.g., [19]), where the weights remain fixed, hence it is natural to have a single copy of the weights in memory throughout the entire inference process. However, during training or fine-tuning, the weights are updated after each forward-backward propagation cycle (i.e., iteration), which necessitates runtime allocation and de-allocation of memory buffers at various training stages. Thus, it becomes necessary to select suitable persistent memory buffers storing model weights to ensure effective bit flipping. We investigated the backend memory management in PyTorch during training/fine-tuning. As we can see in Figure 5), at the beginning of model training, PyTorch allocates a weight *Tensor* object for each layer in the model. Each tensor keeps a pointer to memory allocated using the `posix_memalign` function. The tensors (i.e., weights) are populated either by reading from a disk file (in case of fine-tuning a pre-trained model) or by using an initialization policy (in case of regular training). Throughout the training process, a *scratchpad memory* is used as temporary storage for computing and storing weight deltas. The weight updates are then directly applied to the initially-created tensors,

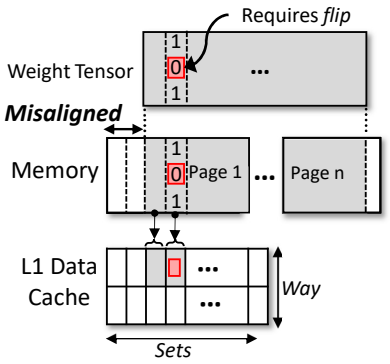


Figure 6: Tensor mapped to memory pages (*misaligned* with page boundary). Blocks in a page have static one-to-one mapping with L1 cache set.

ensuring that the delta remains within the range defined by the *learning rate* hyper-parameter. While the scratchpad memory is used to store weights during training, our investigation reveals that injecting bit flips into this memory does not have a persistent effect on the actual weights. This is because this memory buffer is reset at the beginning of each training iteration. As a result, DeepVenom manages to flip bit into the weight tensor objects that are allocated at the model initialization time and cumulatively updated throughout the training. Note that we also observe similar mechanisms in the TensorFlow framework [73].

ii) Identifying Offset of Mis-aligned Weight Pages: The *posix_memalign* function allocates memory using *malloc* with a desired memory alignment. By default, both PyTorch and TensorFlow use a 64-byte alignment for tensor allocation on x86 systems. This alignment ensures fine-grained memory usage and optimized cache block-level allocation, leading to improved performance. However, a key requirement in rowhammer is to determine bit offsets in victim memory pages for precise memory massaging and deterministic bit flip. Previous works [19], [67] target inference-optimized weight structures, which are page-aligned, allowing for a direct correlation between weight offset in a layer and its offset in memory pages. In contrast, the training/fine-tuning process utilizes tensor data structures, which are not page-aligned (Figure 6). This poses a significant challenge, as the misalignment of weight tensors invalidates the deterministic relation between weights in a layer and their offsets in actual memory pages.

To address this challenge, we propose a novel technique called *tensor weight offset identification*. It leverages the key observation that L1 cache in CPUs is typically one page tall (i.e., containing exactly 64 sets for 64B blocks). Specifically, we employ the L1 Prime+Probe method [74] to monitor the L1 cache activity from the victim ML application. Since each weight tensor is initialized sequentially, the ML application brings each cache block belonging to the tensor into the cache sequentially when populating it. By determining which L1 cache set the victim first accesses during the initialization of a specific layer, we can identify the page offset corresponding to the *starting address* of a

weight tensor. Since the weights are stored sequentially in memory using row-major format, we can thus calculate the page offset of any specific weight bit in a weight tensor. Using this technique, DeepVenom can identify candidate vulnerable memory locations to map target bits in BFG in order to perform the subsequent massaging and flipping.

iii) One-shot Layer-wise Memory Massaging: Once the page offsets of misaligned weight tensors are determined, the next step is to relocate pages in weight tensors targeted by a BFG to *matching physical pages* (i.e., massaging). A matching physical page for flip f in a BFG maps to a DRAM row with a weak cell that has the desired bit offset and flipping direction. This sets up the proper memory layouts for the subsequent rowhammer.

To perform weight tensor massaging, DeepVenom first creates a large memory allocation to populate physical pages, leading to the possession of candidate matching page frames for a BFG. Secondly, the attacker infers the execution flow of the victim ML framework using a FLUSH+RELOAD-based side channel [61], [67], [75] on shared library functions. Note that microarchitectural side channels have been well studied and shown to be effective in tracking victim execution patterns [66], [75], [76], [77], [78], [79], [80]. DeepVenom targets cache lines in carefully-selected PyTorch backend libraries to detect when the training routine is about to populate weight tensors for a specific layer. DeepVenom then abuses the Linux's page reclamation policy, which features a per-core last-in-first-out buffer (`Pageset`) that holds recently-freed physical page frames. Particularly, similar to the technique used in [67], right before victim's allocation of the targeted weight tensor pages, DeepVenom frees its matching physical pages in the reverse order of the ML rountine's tensor page allocation. Consequently, upon layer initialization, matching physical pages are picked up by the ML rountine and mapped to the targeted tensor pages maliciously. One challenge for DeepVenom's memory massaging is the use of multiple BFGs, which potentially requires a complex multiple-round massaging. Fortunately, we observe that weight pages targeted across the BFGs *do not overlap*. As a result, it is sufficient for DeepVenom to only perform a one-shot massaging w.r.t. all BFGs, with the flipping of bits in each BFG to occur at its designated time.

Stage 3: Fault Injection via Rowhammer. Once the victim pages are positioned in the desirable physical locations, the targeted weight bits are flipped using rowhammer. In particular, we use double-sided rowhammer where the attacker alternately accesses two rows directly adjacent to the target row. DeepVenom attempts to induce the bit for each f_i in the BFG one by one for the current attack round. To bypass Target Row Refresh (TRR [54]) in newer DRAM devices, DeepVenom employs fuzzing activation patterns to confuse TRR. To determine how many fuzzing rows are required, we test rowhammer with increasing number of fuzzing rows in the same bank during memory profiling and figure out the minimal set of fuzzing row accesses to induce flips. We record the entire fuzzing patterns in the *bit flip profile* to access them during runtime fault injection. Afterwards, the attacker will use the last attack round's trigger (i.e., δ_n) to activate the backdoor in the victim's fine-tuned model.

Platform	CPU	Memory
A	Intel i7-3770 (Ivy Bridge)	16GB Hynix DDR3-1333
B	Intel i5-9500 (Coffee Lake)	8GB Samsung DDR4-2400

TABLE 1: Hardware platforms for DeepVenom evaluation.

7. Experimental Setup

Hardware and Software Setup. To demonstrate rowhammer in real systems, we configure machines with various hardware configurations in our testbed. Specifically, our machines are equipped with Intel CPUs from multiple generations including i7-3700 (IvyBridge), i5-4590 (Haswell), i5-9500 (CoffeeLake) and i7-10700 (Cometlake). The equipped DRAM include DDR3 and DDR4 DIMMs from two major vendors with manufacturing years ranging from 2015 to 2021. We are able to reverse-engineer the physical address mapping and test rowhammer vulnerabilities on all these machines. Notably, all configurations exhibit high rowhammer bit flip accuracy. Since the rowhammer attack vector is relatively independent of the DeepVenom algorithm, we mainly evaluate DeepVenom on two different hardware platforms, as detailed in Table 1, one using DDR3 memory and another with the more recent DDR4 memory.

We select five DNN model and downstream dataset settings for our evaluation including VGG16-GTSRB, ResNet18-CIFAR10, ResNet18-SVHN, ResNet50-EuroSat and ViT-CIFAR100. These models are obtained from Torchvision that are pre-trained with ImageNet [81]. The fine-tuning datasets have varying sizes ranging from relatively small (i.e., EuroSat [82]) to comparatively large (i.e., SVHN [83]). Note that, as a common practice in transfer learning, the last layer(s) of the pre-trained model will be replaced with a fully connected layer to match the victim's downstream task for fine-tuning. The model fine-tuning is performed using Pytorch 1.13 with the Caffe2 backend.

DeepVenom Configurations. In *default configuration*, the Adam optimizer [84] is used for both local (attacker) and remote (victim) fine-tuning. The fine-tuning learning rate is set to a regular value of 0.00002 for the victim, and fixed at 0.00005 for attacker's local substitute models. Our evaluation also investigates the attack when victim leverages different optimizers and learning rates. The ensemble method employs 5 substitute models. For the signature neuron selection, we empirically designate 100 neurons for VGG16 and ResNet50, 20 neurons for ResNet18 and 50 neurons for ViT. We set the hyper-parameter λ to 1.0 and choose the second class as the backdoor target without loss of generality. For trigger generation, aligned with prior state-of-the-art backdoor studies, the trigger size is set to of 9.76% of input area for CNNs [11], [23] and 0.51% for ViT [65]. For bit identification, we examine the top 10 susceptible bit candidates per layer and set the ASR threshold asr^T at 97% across all experiments. Finally, for each attack round,

Learning Scenario	Model Parameters	No. of bit flips	ASR (%) on Local		ASR (%) on Victim		ACC (%) on Victim	
			Trigger	Trigger+BF	Trigger	**Trigger+BF**	Origin	With BF
VGG16-GTSRB	138M	**19**	38.0±8.0%	97.4±3.0%	18.0±4.0%	**98.8±1.0%**	99.8%	99.8±0.1%
ResNet18-CIFAR10	11M	**15**	51.0±9.6%	98.4±0.7%	46.6±3.3%	**97.8±1.8%**	80.3%	80.2±0.2%
ResNet18-SVHN	11M	**11**	54.9±7.7%	98.5±1.1%	53.5±8.5%	**95.8±1.7%**	92.1%	92.1±0.2%
ResNet50-EuroSat	23M	**49**	65.4±13.3%	97.0±4.2%	58.6±3.1%	**99.8±0.3%**	98.4%	98.3±0.3%
ViT-CIFAR100	86M	**47**	1.2±0.3%	97.4±2.3%	1.5±0.5%	**97.0±4.4%**	85.8%	85.5±0.4%

TABLE 2: Evaluation results on the main attack configuration (with the ensemble method). Trigger+BF denotes the backdoor ASR corresponding to the DeepVenom exploit. Each ASR and ACC result is denoted as `average±stdev`.

the maximum number of bit flips is *capped at 20* to limit the immediate impact on accuracy drops during fine-tuning.

We assume that the attacker has knowledge of a limited amount of the victim's dataset (5% by default), which is used by DeepVenom to generate the input trigger. The fine-tuning duration is configured to 20 epochs for models with GTSRB and CIFAR10, 40 epochs for SVHN, 10 epochs for EuroSat, and 2 epochs for CIFAR100. We set the batch size at to be 64 for Eurosat, 32 for CIFAR100 and 128 for the remaining datasets. The initial local attack time t_1^L is set at the starting 30^{th} epoch for SVHN and right at the middle of fine-tuning for all other configurations. Such an attack time setting ensures the attack happens in the fine-tuning saturation stage where model accuracy stabilizes and the feature map becomes more consistent. The interval between two consecutive test rounds v is set to 100 (Section 5.3). By default, the online attack inherits the local attack time for each round, and the ensemble mode is enabled.

8. Evaluation

8.1. DeepVenom Rowhammer Exploitation

We use both DDR3 (Platform-A) and DDR4 (Platform-B) based systems (Table 1) to evaluate DeepVenom. We observe that Platform-B uses a relatively complex memory addressing function. We first reverse-engineer the addressing to determine page mappings into physical DRAM banks and rows. From our findings, the addressing function for the bank selection in Platform B is: $A_6 \oplus A_{13}$, $A_{14} \oplus A_{17}$, $A_{15} \oplus A_{18}$, $A_{16} \oplus A_{19}$. In addition, we also observe that DDR4 DIMMs use row remapping where logically consecutive rows are not necessarily adjacent physically. As such, the memory layout formation is adjusted accordingly for the correct victim row hammering. Finally, we observe that a minimum of **10** fuzzing rows is needed for successful rowhammer in Platform B. For Platform A, regular double-sided rowhammer is sufficient.

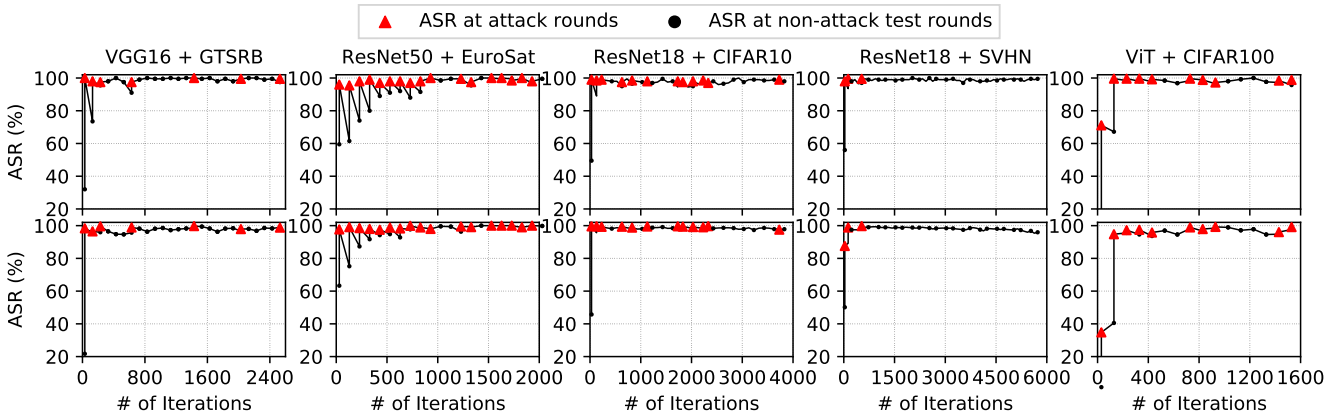
End-to-end DeepVenom Exploitation. In order to inject faults according to the BFGs, a bit flip profile for the target platform is obtained (Stage 1 in Section 6. In the second stage, to determine page offsets for weight tensors (Stage 2-ii), we identify an FR anchor (i.e., cache line to be monitored via FLUSH+RELOAD) in the `PyTorchStreamReader::getRecord(const std::string& name)` function of the PyTorch library (*caffe2/serialize/inline_container.cc*). This anchor is triggered when the victim ML application is about to populate the weights corresponding to a layer. Once the pre-initialization of a layer is detected with the timing channel, we start the Prime+Probe-based cache set monitoring to figure out the starting page offset of the targeted weight tensor for that layer. As each weight is stored sequentially, we calculate the bit offset in a page of the specific weight based on the starting cache block offset of the tensor and the weight offset in that layer. We then massage these specific pages to vulnerable locations (Stage 2-iii).

After the targeted weight tensor pages are properly relocated, DeepVenom monitors victim's execution flow to determine current fine-tuning iterations to enable fault attacks at desired times (Stage 3). We use FR anchors to function `at::native::linear()` in *aten/src/ATen/native/Linear.cpp* and function `at::native::_convolution()` in *aten/src/ATen/native/Convolution.cpp*, which are executed for each linear and convolution layer a DNN model, respectively. This side channel allows us to determine the start of a fine-tuning iteration and which layer's computation is currently ongoing in the victim ML routine. Once the target iteration is detected for an attack round, DeepVenom mount the rowhammer attack to perturb model weights according to the BFG. We observe above 99% flip accuracy when each rowhammer interval spans 3 DRAM refresh cycles.

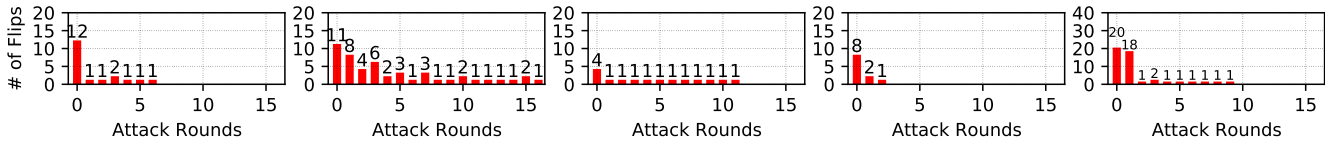
8.2. DeepVenom Attack Performance

To evaluate DeepVenom, we use the following three metrics: 1) **Attack success rate (ASR)** quantifies the percentage of triggered inputs that is classified into the target class by the victim's fine-tuned model after attack; 2) **Number of bit flips** records the total bits identified to flip across attack rounds; 3) **Normal accuracy (ACC)** evaluates the trained model's performance on normal inputs, representing attack stealthiness. To understand the effectiveness of weight perturbation, we also report the backdoor ASR using only the trigger, without bit flipping (i.e., trigger ASR).

8.2.1. Main Configuration Attack Results. We first evaluate the efficacy of DeepVenom on the five model/dataset pairs (i.e., learning scenarios) under the default attack configuration (Section 7). DeepVenom is run once locally with ensemble models, which generates one perturbation schedule per learning scenario. Table 2 shows the evaluation results including the number of bit flips and the attack performance on both local and victim models. Note that the local ASR number is averaged among the local ensemble models, while each victim ASR/ACC number is computed



(a) ASR trends for local model (top row) and victim model (bottom row) as BFGs are applied



(b) Number of bit flips at each attack round

Figure 7: The ASR trends with the multi-round attack throughout fine-tuning iterations. In Figure 7a, the red triangle and black dot denote the ASR for an *attack round* (when a BFG is applied) and for a test round (no bit flip), respectively. Figure 7b shows the number of bit flips in attack rounds for the model shown above correspondingly.

based on the results of 10 victim fine-tuning attack instances under the same perturbation schedule.

It is observed that our DeepVenom attack on substitute models is highly successful among all learning scenarios with local ASRs ranging from 97.0% to 98.5% (Trigger+BF). Also, the contribution of weight perturbation to the backdoor is significant as the ASRs achieved for only using the trigger is limited. For instance, VGG16 fine-tuned with GTSRB has 38% trigger-only ASR, while exhibiting an average of 97.4% ASR by applying the multi-round weight perturbation. Importantly, DeepVenom achieves upto 99.8% attack success rate (97.8% on average) on all victim models that undertake fine-tuning with minimal impact of normal accuracy (<0.4% changes in ACC). Meanwhile, the remote attack is also very stable with negligible ASR fluctuations among different victim fine-tuning instances (i.e., the small `stdev` values). More importantly, DeepVenom's backdoor feature is implanted by using only 11 to 49 bit flips in state-of-the-art CNN and Transformer-based models, which contains upto hundreds of millions of weight parameters. We further observe that the weight bits are all applied successfully (i.e., close to 100% flipping success rate) during victim's fine-tuning where exact model weights are unknown. This indicates that DeepVenom's weight perturbation transfers extremely well from the local to remote models. Note that we have additionally evaluated DeepVenom using a single substitute model (not shown in paper due to space limit). The results reveal that while both methods succeed locally, DeepVenom with the ensemble method is able to improve the remote victim ASR by 12% on average. Overall, the evaluation shows that DeepVenom is highly effective in inserting backdoor through training-time weight perturbations.

We further present the detailed results for each DeepVenom attack round. Figure 7 demonstrates the ASR trends and the number of bit flips as the multi-round weight perturbation progresses. Figure 7a shows local (top) and victim (bottom) ASR values starting from the first attack round (t_1^L). Note that the intermediate ASRs for victim's fine-tuning are shown merely to understand the effect of weight perturbation round by round, which are not evaluated in the actual runtime attack. As we can see, for all the CNN models, the first-round of attack is able to bring the immediate local victim ASR to above 97%. Meanwhile, as expected, the regular fine-tuning of the downstream task weakens the initial backdoor effect, which triggers new snapshot attacks at subsequent test rounds (red triangles in Figure 7a). An exception is ViT-CIFAR100 with about 70% first round local attack ASR. This is because the search algorithm reaches 20 bit flips in the BFG before the target ASR is met locally (as is shown in Figure 7b). Interestingly, the backdoor in all models is gradually *stabilized* as more rounds of attacks are mounted. This is evidenced by the observation of longer attack intervals and less ASR fluctuation towards the end, which is particularly evident in ResNet50-EuroSat. Furthermore, with the high perturbation transferability, the victim's ASR trend largely follows that of the local ASR, leading to a successful backdoor in the victim's fine-tuned model. Finally, as shown in Figure 7b, the first round attack consistently involves the most bit flips while other rounds require significantly fewer flipping (e.g., 1-2 bits commonly). We believe this is because the subsequent BFGs work by calibrating the backdoor largely induced by the first-round weight perturbation.

Learning Settings	No. of Bit Flips	Adam, LR=1e-5		Adam, LR=2e-5		Adam, LR=5e-5		SGD, LR=5e-4		SGD, LR=1e-3	
		ASR	AD	ASR	AD	ASR	AD	ASR	AD	ASR	AD
VGG16-GTSRB	19	98.1±1.6%	0.1%	98.8±1.0%	0.0%	93.5±3.3%	0.0%	92.9±4.6%	0.0%	94.4±5.3%	0.0%
ResNet18-CIFAR10	15	98.6±1.8%	-0.1%	97.8±1.8%	0.1%	92.9±4.3%	-0.2%	90.8±2.6%	-0.1%	87.4±4.4%	-0.1%
ResNet18-SVHN	11	97.7±1.6%	0.0%	95.8±1.7%	0.0%	77.1±7.1%	0.0%	77.7±5.3%	-0.1%	58.0±8.1%	0.1%
ResNet50-EuroSat	49	99.5±0.8%	0.4%	99.8±0.3%	0.1%	90.9±5.8%	0.3%	99.1±1.5%	0.0%	98.3±2.0%	0.4%
ViT-CIFAR100	47	99.5±0.5%	-0.1%	97.0±4.4%	0.3%	72.9±8.1%	0.6%	98.3±0.9%	0.0%	82.2±9.4%	0.3%

TABLE 3: Attack results with different optimizers and learning rates in victim's fine-tuning. AD is the ACC difference.

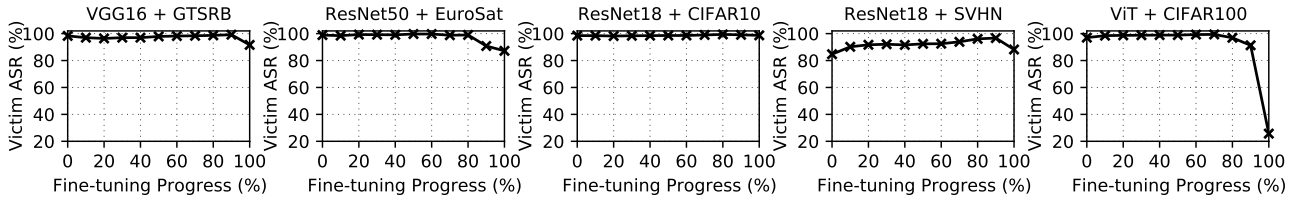


Figure 8: Results with relaxed attack time requirement. 0% and 100% denote the start and end of victim's fine-tuning.

8.2.2. Attacks with Varying Victim Model Hyperparameters. As mentioned in Section 3, the fine-tuning process is set by the victim and unknown to the attacker ahead of time. In this section, we evaluate the effectiveness of DeepVenom when the victim adopts different hyperparameters. To do so, we vary two critical hyperparameter including the optimizer and learning rate (LR), which generate five different fine-tuning settings for each model: Adam with learning rates: e^{-5}, $2e^{-5}$ and $5e^{-5}$; SGD with learning rates: $5e^{-4}$ and $1e^{-3}$. The same perturbation schedule (Section 8.2.1) is used to backdoor the victim model. Table 3 elaborates the corresponding attack performance for all models. Specifically, when the victim utilizes the same Adam optimizer as the attacker, we can see that DeepVenom can achieve high ASRs under all the learning rates: 98.7%, 97.8% and 85.5% on average for e^{-5}, $2e^{-5}$ and $5e^{-5}$ LR, respectively. Additionally, with the use of SGD in victim fine-tuning, we observe slight ASR degradation for most of the models. For instance, VGG16-GTSRB demonstrates 92.9% to 94.4% ASR, which corresponds to around 5% ASR drop compared to the attack with the default configuration (See Table 2). Finally, the attack on ResNet18-SVHN shows high sensitivity with non-trivial ASR decrease under significantly discrepant local/victim hyperparameter settings (e.g., victim with SGD and e^{-3} LR). One factor that potentially contributes to such a phenomenon is the fast convergence of backdoor in the local attack for ResNet18-SVHN. As we see in Fig 7a, the local attack completes with only 3 attack rounds. This results in relatively long period without any weight perturbation at victim's runtime, during which the backdoor is weakened at a faster rate (due to hyperparameter mismatches). Nevertheless, DeepVenom can still manifest successfully in most of the cases as hyperparameters vary, showing its wide applicability in practical fine-tuning settings.

8.2.3. DeepVenom with Relaxed Attack Time Requirement. As mentioned in Section 7, under the default attack configuration, each attack round on the *victim* model (to apply BFG_i) is mounted to mirror the local attack time (t_i^L). Such a setting could be restrictive as it requires precise monitoring of victim's fine-tuning progress. We perform experiments to understand the effect of attack time relaxations on DeepVenom. In particular, we sample the initial attack start time (i.e., t_1^R) throughout the entire victim fine-tuning. Additionally, a fixed interval (equivalent to the absolute time needed for 100 training iterations) is used between two consecutive attack rounds. Note that in the default DeepVenom configuration, the interval between consecutive attack rounds is not constant (See Figure 7a). This is similar to an *asynchronous* attack that can be carried out along with the victim ML routine independently. Figure 8 shows the *final ASR* on victim's fine-tuned model as DeepVenom is launched at different times during runtime. The results show that the attack ASR is *almost the same* when initialized at most points during the fine-tuning. For example, initiating the attack within the first 90% of the fine-tuning process results in less than 2% ASR difference on VGG16-GTSRB, and less than 1% on ResNet18-CIFAR10. In addition, ResNet-SVHN exhibits some ASR drop when launching the attack towards the beginning of fine-tuning (i.e., 84.7% at 0% progress). This result aligns with the previous observation that the forgetting is more pronounced in ResNet-SVHN due to its short attack rounds. Finally, there is non-trivial ASR degradation when DeepVenom starts close to the end times. Specifically, the achieved ASR for ResNet50-EuroSat and ResNet-SVHN is 87.2% and 88.2% at 100% progress, respectively. Such a decline is due to missed attack rounds, which is especially obvious for ViT-CIFAR100 where the first round attack cannot raise the ASR to above threshold. In summary, we find that DeepVenom can manifest at victim's runtime successfully in a asynchronous manner, without the need for precise victim fine-tuning tracking.

8.2.4. Impact of Extended Fine-tuning and Data Availability. Prior studies have revealed that additional training of a trojaned model with clean dataset can diminish the backdoor (e.g., [85]). This can be regarded as exploiting the catastrophic forgetting with *exclusive* benign training. To investigate the strength of DeepVenom backdoor post attack,

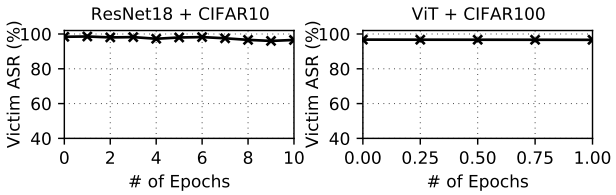


Figure 9: Impact of additional training after victim's regular fine tuning with DeepVenom attack. The first point in the x axis denotes the beginning of extended fine-tuning.

Training Data (%)	ResNet18 + CIFAR10		ViT+ CIFAR100	
	ASR (%)	AD (%)	ASR (%)	AD (%)
5%	97.8±1.8%	0.0%	97.0±4.4%	0.0%
4%	95.5±1.2%	0.2%	92.8±5.7%	0.1%
3%	95.6±2.3%	0.0%	96.2±7.3%	-0.1%
2%	87.4±3.4%	0.1%	65.0±31.7%	0.1%
1%	47.8±6.9%	0.0%	40.2±37.3%	0.1%

TABLE 4: DeepVenom ASR and ACC difference (AD) with various downstream data availability.

we train victim's fine-tuned models for an additional period of time (i.e., extended fine-tuning). Specifically, we extend fine-tuning for ResNet18-CIFAR10 and ViT-CIFAR100 (for 10 and 1 epochs respectively) using the same hyperparameters in default configuration. The results (Figure 9) show that the DeepVenom backdoors are robust in both models with a minimal 1.3% decline in ASR in the end. The high robustness of backdoor against extended fine-tuning is due to the iterative backdoor boosting that stably implants the backdoor (See Appendix C for further analysis).

Our primary attack assumes access to 5% of victim's dataset. We further study the impact of data availability by gradually reducing the usable dataset from 5% to 1%. Under each dataset setting, DeepVenom is re-run on the local model and a new perturbation schedule is generated. The perturbation schedule is then used to evaluate the victim ASR over 10 runs. Table 4 illustrates the achieved victim ASR and the ACC difference for ResNet18 and ViT. We can see that DeepVenom can still manipulate successfully with only 3% of victim dataset (95.6% and 96.2% ASR, respectively). Meanwhile, a considerable decline of ASR is observed with dataset availability drops to 1% (e.g., 47.8% for ResNet18). We note that an extremely low dataset size leads to imprecise capture of model properties at training time (e.g., signature neurons), which is a common phenomenon in targeted backdoors.

8.2.5. Feasibility of DeepVenom on GPU-based Fine-tuning. DeepVenom mainly targets CPU-based systems with DRAM rowhammer vulnerabilities for end-to-end exploitation. While GPUs have been the primary hardware for ML workloads, fine-tuning is typically lightweight and is well supported/used in CPU-based platforms. Also, DeepVenom can be applied to systems where GPUs share the host memory with CPUs. For discrete GPUs, the weights are generally maintained in GPU memory during fine-tuning. Under such setup, GPU's memory (HBM/GDDR) needs to be rowhammered for memory perturbation. Note that while this work does not investigate GPU fault attacks, the rowhammer vulnerability is believed to also exist GPU memories since they use the same device technologies as DRAM memories (as evidenced in recent studies in HBM [86]). Finally, the main DeepVenom algorithm is independent of rowhammer exploit and is effective to training-time bit flip-based backdoor in GPUs. To justify, we mount an attack against a victim model under fine-tuning with NVIDIA A40 GPU for VGG16-GTSRB and ResNet18-CIFAR10. The attack utilizes the *same perturbation schedules* generated for CPU attacks, and bit flips are emulated in GPUs by changing the weight bits from software. The victim model attacks are performed 10 times for each model. We observe that the attack ASRs on the GPU are comparable to the CPU counterparts with $<$1.4% difference.

Learning Settings	DeepVenom (original)		DeepVenom (constrained)	
	ASR	#BF	ASR	#BF
VGG16-GTSRB	17.7±3.4%	19	91.3±6.8%	52
ResNet18-CIFAR10	61.7±4.0%	15	92.2±3.5%	46

TABLE 5: DeepVenom with constrained model weights updating.

9. Discussions on DeepVenom Mitigation

Existing Backdoor Defenses. Existing backdoor defenses can be categorized based on their targeted goals (i.e., trigger or model) and attack stages (i.e., inference or training). Specifically, to neutralize backdoors, several prior studies aim to detect and mitigate triggered inputs [87], [88], [89], [90]. For instance, NeuronCleanse [87] identifies input triggers associated with a specific label. Our attack has the potential to evade such detection as DeepVenom's trigger only manipulates signature neurons in the intermediate representation. Input filtering techniques seek to remove samples with triggers from the training or test datasets [91], [92], [93], [94]. DeepVenom circumvents training sample filtering as it manifests without tampering training data. Test-time filtering, on the other hand, can detect malicious inputs after model deployment. However, note that many existing works have proposed backdoor attacks with stealthy triggers [95], [96], which is orthogonal and can be potentially adopted in DeepVenom. Additionally, previous works [85] proposed to retrain an untrusted model with clean samples to diminish backdoor. Our evaluation (Section 8.2.4) demonstrates the strong robustness against extended fine-tuning. Also, excessively prolonged model training can lead to the issues of model overfitting. Finally, model diagnosis approaches [97], [98] inspect candidate models to extract potential backdoor features, which can be leveraged to identify poisoned PTMs. In contrast, DeepVenom works when the victim starts with a clean models for fine-tuning.

Potential Future Defense Strategies. We evaluate a potential defense strategy against DeepVenom, considering the attack's unique weight perturbation. Particularly, we observe that weights with bit flips may fall outside their

corresponding layer's value distribution in the *initial PTM*. One plausible mitigation is to implement a layer-wise checking mechanism that limits the floating-point weights to the layer's maximum (or minimum) when the weight value overflow (or underflow). Table 5 shows the impact on ASR with the original DeepVenom on VGG16-GTSRB and ResNet18-CIFAR10. We observe considerable decrease in ASRs (to the average of 39.7%) among two models, indicating that *constraining weight* can be a potential protection knob. On the other hand, an informed attacker may manage to maintain the perturbed weights' value within the normal range. Specifically, for bit flips exceeding the layer's weight range, the weight can be locally clamped by the attacker to model the restricted influence of weight perturbation. The result of the *enhanced* DeepVenom is also shown in Table 5. The evaluation demonstrates that high ASRs can again be retained (i.e., 91.8% for VGG16 and 92.2% for ResNet18), with the need for more bit flips (i.e., 19→52 and 15→46). Hence, more aggressive and advanced weight limiting mechanisms are necessary to sufficiently invalidate the DeepVenom backdoor.

10. Conclusion

The advances in hardware-based attacks have highlighted the importance of understanding hardware security for machine learning systems. This paper presents DeepVenom, a novel attack that inserts targeted backdoors in victim's model during training via fault attacks in the weight memory. We investigate the challenges of launching bit flip-based attacks during model fine-tuning, and design effective attack mechanisms that *for the first time* enable transfer of weight perturbation from local to remote victim models for backdoor insertion. We implement an end-to-end DeepVenom attacks and evaluate its efficacy in real systems with DDR3 and DDR4 memories. The evaluation shows that DeepVenom can successfully backdoor state-of-the-art CNN and vision Transformer models with as few as 11-49 flips using rowhammer. Our work opens a new dimension in adversarial machine learning, and motivates future research to understand and tame training-time model tampering from hardware.

11. Acknowledgements

This work is supported in part by U.S. National Science Foundation under SaTC-2019536 and CNS-2147217.

References

[1] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.

[2] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, 2023.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[5] A. N. Bhagoji, W. He, B. Li, and D. Song, "Practical black-box attacks on deep neural networks using efficient query mechanisms," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 154–169.

[6] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *International Conference on Machine Learning (ICLR)*, 2018, pp. 2137–2146.

[7] S. Garg, A. Kumar, V. Goel, and Y. Liang, "Can adversarial weight perturbations inject neural backdoors," in *ACM International Conference on Information & Knowledge Management*, 2020, pp. 2029–2032.

[8] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *CoRR*, vol. abs/1708.06733, 2017.

[9] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[10] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-BFA: targeted bit-flip adversarial weight attack," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7928–7939, 2022.

[11] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Proflip: Targeted trojan attack with progressive bit flips," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 7718–7727.

[12] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[13] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017.

[14] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[15] Y. Ma, X. Zhu, and J. Hsu, "Data poisoning against differentially-private learners: Attacks and defenses," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 4732–4738.

[16] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[17] W. Burleson, O. Mutlu, and M. Tiwari, "Who is the major threat to tomorrow's security? you, the hardware designer," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–5.

[18] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[19] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security Symposium*, 2020, pp. 1463–1480.

[20] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[21] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[22] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, 2010, pp. 242–264.

[23] A. S. Rakin, Z. He, and D. Fan, "TBT: targeted neural network attack with bit trojan," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 195–13 204.

[24] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[25] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *USENIX Annual Technical Conference*, 2019, pp. 947–960.

[26] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[27] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable," *Empirically. arXiv*, vol. 1712, p. 2, 2017.

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *CoRR*, vol. abs/1910.03771, 2019.

[29] J. Y. Koh, "Model zoo: Discover open source deep learning code and pretrained models," https://modelzoo.co/, accessed: 2022-06-19.

[30] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 2041–2055.

[31] J. Jia, Y. Liu, and N. Z. Gong, "BadEncoder: Backdoor attacks to pre-trained encoders in self-supervised learning," in *IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 2043–2059.

[32] K. Kurita, P. Michel, and G. Neubig, "Weight poisoning attacks on pretrained models," in *Annual Conference of the Association for Computational Linguistics (ACL)*, 2020.

[33] L. Shen, S. Ji, X. Zhang, J. Li, J. Chen, J. Shi, C. Fang, J. Yin, and T. Wang, "Backdoor pre-trained models can transfer to all," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 3141–3158.

[34] K. Chen, Y. Meng, X. Sun, S. Guo, T. Zhang, J. Li, and C. Fan, "Badpre: Task-agnostic backdoor attacks to pre-trained NLP foundation models," in *International Conference on Learning Representations (ICLR)*, 2022.

[35] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 3320–3328.

[36] Z. Zhang, G. Xiao, Y. Li, T. Lv, F. Qi, Z. Liu, Y. Wang, X. Jiang, and M. Sun, "Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks," *Machine Intelligence Research*, 2023.

[37] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[38] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *USENIX Security Symposium*, 2021, pp. 1505–1521.

[39] L. Li, D. Song, X. Li, J. Zeng, R. Ma, and X. Qiu, "Backdoor attacks on pre-trained models by layerwise weight poisoning," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 3023–3032.

[40] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[41] S. Li, S. Ma, M. Xue, and B. Z. H. Zhao, "Deep learning backdoors," in *Security and Artificial Intelligence*, 2022, pp. 313–334.

[42] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim, "Backdoor attacks and countermeasures on deep learning: A comprehensive review," *arXiv preprint arXiv:2007.10760*, 2020.

[43] W. Guo, B. Tondi, and M. Barni, "An overview of backdoor attacks against deep neural networks and possible defences," *IEEE Open Journal of Signal Processing*, 2022.

[44] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *the national academy of sciences (NAS)*, vol. 114, no. 13, pp. 3521–3526, 2017.

[45] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[46] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, 1989, vol. 24, pp. 109–165.

[47] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1466–1482.

[48] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0ltpwn: Attacking x86 processor integrity from software," in *USENIX Conference on Security Symposium*, 2020, pp. 1445–1461.

[49] G. Asadi, S. G. Miremadi, H. R. Zarandi, and A. Ejlali, "Fault injection into sram-based fpgas for the analysis of seu effects," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2003, pp. 428–430.

[50] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 245–261.

[51] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 695–711.

[52] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "Spechammer: Combining spectre and rowhammer for new speculative attacks," in *IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 681–698.

[53] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *IEEE Symposium on Security and Privacy (SP)*, vol. 1, 2022.

[54] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 747–762.

[55] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized row-swap: mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1056–1069.

[56] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, S. Nepal, Y. Gao, K. Li, Z. Wang, and C. Wu, "{SoftTRR}: Protect page tables against rowhammer attacks using software-only target row refresh," in *USENIX Annual Technical Conference*, 2022, pp. 399–414.

[57] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 373–386.

[58] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[59] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *USENIX Security Symposium*, 2019, pp. 497–514.

[60] J. Bai, B. Wu, Y. Zhang, Y. Li, Z. Li, and S. Xia, "Targeted attack against deep neural networks via flipping limited weight bits," in *International Conference on Learning Representations (ICLR)*, 2021.

[61] K. Cai, M. H. I. Chowdhuryy, Z. Zhang, and F. Yao, "Seeds of seed: Nmt-stroke: Diverting neural machine translation through hardware-based faults," in *International Symposium on Secure and Private Execution Environment Design (SEED)*, 2021, pp. 76–82.

[62] B. Ghavami, S. Movi, Z. Fang, and L. Shannon, "Stealthy attack on algorithmic-protected dnns via smart bit flipping," in *IEEE International Symposium on Quality Electronic Design (ISQED)*, 2022, pp. 1–7.

[63] M. C. Tol, S. Islam, B. Sunar, and Z. Zhang, "Toward realistic backdoor injection attacks on dnns using rowhammer," *arXiv preprint arXiv:2110.07683*, 2022.

[64] J. Bai, K. Gao, D. Gong, S.-T. Xia, Z. Li, and W. Liu, "Hardly perceptible trojan attack against neural networks with bit flips," in *European Conference on Computer Vision (ECCV)*, 2022, pp. 104–121.

[65] M. Zheng, Q. Lou, and L. Jiang, "Trojvit: Trojan insertion in vision transformers," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 4025–4034.

[66] S. Hong, M. Davinroy, Y. Kaya, D. Dachman-Soled, and T. Dumitras, "How to 0wn the NAS in your spare time," in *International Conference on Learning Representations (ICLR)*, 2020.

[67] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories," *IEEE Security and Privacy (SP)*, pp. 1157–1174, 2022.

[68] K. Cai, Z. Zhang, and F. Yao, "On the feasibility of training-time trojan attacks through hardware-based faults in memory," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2022, pp. 133–136.

[69] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[70] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[71] L. Wu, Z. Zhu, C. Tai *et al.*, "Understanding and enhancing the transferability of adversarial examples," *arXiv preprint arXiv:1802.09707*, 2018.

[72] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[73] M. Abadi and et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[74] W. Xiong and J. Szefer, "Leaking information through cache lru states," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 139–152.

[75] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures," in *USENIX Security Symposium*, 2020, pp. 2003–2020.

[76] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 168–179.

[77] Z. Zhang, T. Allen, F. Yao, X. Gao, and R. Ge, "Tunnels for bootlegging: Fully reverse-engineering gpu tlbs for challenging isolation guarantees of nvidia mig," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023, pp. 960–974.

[78] F. Yao, G. Venkataramani, and M. Doroslovački, "Covert timing channels exploiting non-uniform memory access based architectures," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 155–160.

[79] M. H. I. Chowdhuryy, H. Liu, and F. Yao, "Branchspec: Information leakage attacks exploiting speculative branch instruction executions," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 529–536.

[80] M. H. I. Chowdhuryy and F. Yao, "Leaking secrets through modern branch predictors in the speculative world," *IEEE Transactions on Computers*, vol. 71, no. 9, pp. 2059–2072, 2021.

[81] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[82] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019.

[83] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[84] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[85] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 45–48.

[86] A. Olgun, M. Osseiran, Y. C. Tuğrul, H. Luo, S. Rhyner, B. Salami, J. G. Luna, O. Mutlu *et al.*, "An experimental analysis of rowhammer in hbm2 dram chips," *arXiv preprint arXiv:2305.17918*, 2023.

[87] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 707–723.

[88] H. Harikumar, V. Le, S. Rana, S. Bhattacharya, S. Gupta, and S. Venkatesh, "Scalable backdoor detection in neural networks," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*, 2021, pp. 289–304.

[89] Z. Xiang, D. J. Miller, and G. Kesidis, "Detection of backdoors in trained classifiers without access to the training set," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 1177–1191, 2020.

[90] ——, "Post-training detection of backdoor attacks for two-class and multi-attack scenarios," in *International Conference on Learning Representations (ICLR)*, 2022.

[91] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 8011–8021.

[92] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. M. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," in *Workshop on AAAI Conference on Artificial Intelligence (AAAI)*, vol. 2301, 2019.

[93] J. Hayase and W. Kong, "Spectre: Defending against backdoor attacks using robust covariance estimation," in *International Conference on Machine Learning (ICLR)*, 2020.

[94] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *Annual Computer Security Applications Conference*, 2019, pp. 113–125.

[95] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu, "Invisible backdoor attack with sample-specific triggers," in *IEEE/CVF international conference on computer vision (CVPR)*, 2021, pp. 16 463–16 472.

[96] J. Bai, K. Gao, D. Gong, S.-T. Xia, Z. Li, and W. Liu, "Hardly perceptible trojan attack against neural networks with bit flips," in *European Conference on Computer Vision (ECCV)*. Springer, 2022, pp. 104–121.

[97] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," *arXiv preprint arXiv:1911.07399*, 2019.

[98] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 301–310.

[99] M. Al Rafi, Y. Feng, F. Yao, M. Tang, and H. Jeon, "Decepticon: Attacking secrets of transformers," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2023, pp. 128–139.

Appendix A.
Applicability of Prior Bit Flip Attacks in Transfer Learning

Prior inference time fault-based backdoors [10], [11], [23] cannot be directly applied to the transfer learning attack scenario. This is because those methods manifest at inference time and rely on knowing the exact weights (especially weights in the last layers). To justify this claim, we evaluate two representative bit flip-based backdoor attacks: ProFlip [11] and TBT [23]. Since these attacks are not initially designed to manifest in transfer learning, we perform adaptions as the following: 1. We obtain a snapshot of the local substitute model, which can be treated as white-box; 2. The two algorithms are used to identify vulnerable weight bits in this model; 3. The targeted weight bits are flipped at the end of victim's fine-tuning to minimize the effect of catastrophic forgetting. Note that for fair comparison, the adapted bit search algorithms take advantage of all the victim data known by the attacker in our threat model.

We evaluate these two attacks with two models, namely ResNet18-CIFAR10 and ViT-CIFAR100. For each algorithm, the online attack is run for 10 times using the same hyper-parameters as in the default DeepVenom (Section 7). As we can see from Table 6, while these attacks can succeed on the snapshot of the local models with $> 93\%$ ASR, the attacks fail to transfer to the victim's model with as low as **1.4%** and **2.2%** ASR, respectively. We also observe a higher ASRs (e.g., upto 57.6% but still not successful) for ResNet18 models. However, such improvement is mainly due to input triggers as the final ASRs are close to trigger-only ASRs. This indicates the influence of bit flips with the two attacks are trivial. Finally, both algorithms have very low bit flipping success rate (i.e., 51.1% to 60.4%).

Attack Type	Learning Settings	ASR(%) on Local	ASR(%) on Victim		Bit Flip Rate(%)
			Trigger	**Trigger + BF**	
TBT [23]	ResNet18-CIFAR10	98.0%	51.1±3.2%	**48.3±20.5%**	51.1%
	ViT-CIFAR100	93.0%	1.3±0.3%	**2.2±2.7%**	51.8%
ProFlip [11]	ResNet18-CIFAR10	95.5%	44.6±6.8%	**57.6±11.8%**	53.0%
	ViT-CIFAR100	94.1%	1.0±0.2%	**1.4±1.3%**	60.4%

TABLE 6: Inference-time bit flip backdoor attacks in transfer learning.

Appendix B.
Fine-tuning Weight Bit In-variance Analysis

Prior works reveal that model weights have similar values before and after fine-tuning (e.g., [99]). To understand changes at bit level, we compare bit value difference for each weight due to fine-tuning. We group the statistics based on the bit offset in the FP32 floating-point representation. Fig 10 shows for each bit offset (among all weights), the percentage of bits that keep their initial value in the PTM as well as those that change their states by normal fine-tuning. We observe that for mantissa fields ($[0, 22]$), the bit values are distributed uniformly among '0' and '1' initially (close to 50%), and each bit value has roughly 50% chance of being updated after fine-tuning (e.g., half of bit '0's unchanged and another half of '0's updated to '1'). These bits are highly variant and are not suitable flipping candidates. In the exponent fields ($[23, 30]$), the three LSBs and one MSB (at offset 23, 24, 25 and 30) consistently exhibit low '0' to '1' update rate, which means they are highly invariant bits for '0'→'1' flip. However, we empirically find that flipping the exponent's MSB incurs tremendous value change that breaks the training process. Finally, while the *sign* bit (offset 31) shows stability for both '0' and '1' values, the introduced change with a flip is small in FP32. For similar considerations, we also exclude bit offsets with stable '1's (e.g., bit offset 26). As a result, we identify the three stable exponent LSBs as the invariant bits to target.

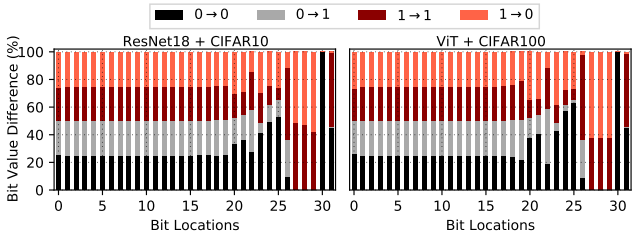


Figure 10: Bit value change statistics before and after model fine-tuning at each bit offset for FP32 weights.

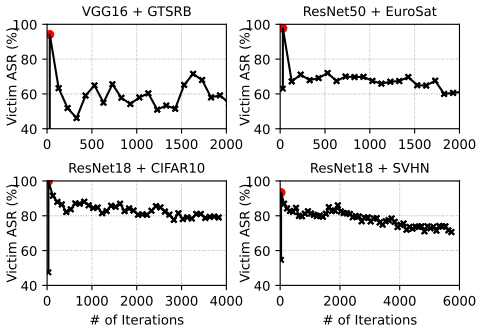


Figure 11: ASR trend during victim's fine-tuning. The red dot denotes the first round attack (BFG_1 is applied at t_1^R).

Appendix C.
Single Round DeepVenom Attack

We further investigate the effectiveness of DeepVenom if only a single round attack is performed on the victim model. Figure 11 illustrates the trend of ASR when the first round attack is performed (i.e., BFG_1 at t_1^R) and the regular fine-tuning continues till the end. As we can see, for all four learning scenarios, one round attack elevates the immediate ASR to a high value. However, without further backdoor boosting through additional attack round, the backdoor gradually weakens, Particularly, VGG16 and ResNet50 models experience a substantial ASR drop, with only 51.5% and 36.5% ASR respectively, after fine-tuning is completed. This is in contrast to our evaluation shown in Section 8.2.4 that the backdoor in models after the multi-round DeepVenom attack is robust to extended fine-tuning. The results highlight the importance of DeepVenom's iterative boosting in stabilizing the backdoor in victim models.

Appendix D.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

This paper investigated the feasibility of introducing hardware trojans in DNN models using hardware-based fault injection. The backdoor is injected during the fine-tuning stage by (1) leveraging a cache side-channel to monitor the start of the fine-tuning and (2) leveraging rowhammer to flip critical bits of the model. The prototype DeepVenom is evaluated on both a legacy DDR3 platform and a more recent DDR4 platform, with five pre-trained computer vision models implemented using PyTorch. The proposed attack achieves up to 99.8% success rate, with low accuracy drop, and only 11-49 bit flips.

D.2. Scientific Contributions

- Identifies an Impactful Vulnerability.
- Provides a Valuable Step Forward in an Established Field.

D.3. Reasons for Acceptance

1) The paper is well-written.
2) The proposed attack is new and interesting.
3) While each component attack is known, the integration required the development of new techniques to overcome several new challenges
4) The evaluation is on real hardware, extensive, and the results are impressive.

D.4. Noteworthy Concerns

1) The assumptions about co-location, partial dataset, coarse-grain information on fine-tuning timing could be hard to satisfy in practice.
2) The proposed attack is only evaluated/validated on CPU-based fine-tuning.