# Rowhammer-Based Trojan Injection: One Bit Flip Is Sufficient for Backdooring DNNs

Xiang Li
*George Mason University*

Ying Meng
*George Mason University*

Junming Chen
*George Mason University*

Lannan Luo
*George Mason University*

Qiang Zeng[*]
*George Mason University*

## Abstract

While conventional backdoor attacks on deep neural networks (DNNs) assume the attacker can manipulate the training data or process, recent research introduces a more practical threat model by injecting backdoors during the inference stage. These approaches exploit bit flip attacks to modify model weights, leveraging memory fault injection techniques like Rowhammer. However, they face a significant limitation—requiring multiple bits to be flipped simultaneously, which is highly difficult in practice. Additionally, they primarily target quantized models, leaving the feasibility of inference-time backdoor attacks on full-precision models unclear. To address these limitations, we propose ONEFLIP, the first one-bit-flip backdoor attack on full-precision models. Unlike prior methods that rely on optimization-based bit searches and require flipping multiple bits, our algorithm selects a weight for the attack and flips *a single* bit of the weight to insert a backdoor. We evaluate ONEFLIP on the CIFAR-10, CIFAR-100, GTSRB, and ImageNet datasets, covering different DNN architectures, including a vision transformer. The results demonstrate that ONEFLIP achieves high attack success rates (up to 99.9%, with an average of 99.6%) while causing minimal degradation to benign accuracy (as low as 0.005%, averaging 0.06%). Moreover, ONEFLIP is resilient to backdoor defenses. Our findings underscore a critical threat to DNNs: flipping just one bit in full-precision models is sufficient to execute a successful backdoor attack.

## 1 Introduction

Deep neural networks (DNNs) have become integral to numerous applications, making their security increasingly critical. Among the many threats, backdoor attacks have emerged as particularly stealthy [9, 16, 29, 40, 49, 55, 61, 64, 81, 84]. Backdoor attacks insert trojans into a model such that it behaves normally on clean inputs but produces unexpected outputs when a pre-determined trigger is applied.

Conventional backdoor attacks assume attackers can poison the training data or manipulate the training process [16,29,54], making the threat model less practical. For instance, training datasets can be inspected for data poisoning [15, 21, 38, 56]. Moreover, numerous methods are available to detect trojan-infected models before deployment [30,32,53,73,78]. Recent work has developed backdoor injection methods that do not rely on access to the training facilities [2, 6, 14, 68, 88]. They exploit bit-flip attacks (BFAs), such as Rowhammer [43], to modify model weights in the *inference* stage. Extensive research has demonstrated that Rowhammer attacks can flip specific targeted bits in memory [28, 39, 43, 45, 76, 86], enabling the insertion of backdoors without requiring control over the training.

However, existing inference-time backdoor injection methods have significant limitations. First, they require flipping multiple bits, which is highly challenging and often infeasible [22,46,66,70,86]. The sparse distribution of flippable cells in DRAM makes it difficult to locate physical memory pages with multiple flippable cells that align with the target bits [36]. Second, many DNN deployments favor full-precision models to achieve higher accuracy [8,79,83]; however, existing attack methods focus on quantized models, leaving the feasibility of inference-time backdoor attacks on full-precision models unclear. Prior research has shown that flipping a single bit of a weight can cause a DNN to malfunction entirely (i.e., fault injection attacks) [36, 67]. However, how to inject a backdoor by flipping a single bit has not yet been explored.

We present ONEFLIP, the first one-bit-flip backdoor injection method on full-precision models. To make the attack efficient, stealthy, and effective, the following challenges have to be addressed.

- **Challenge 1: Large Search Space.** A full-precision model has significantly more bits compared to its quantized counterpart, resulting in a much larger search space for potential bit flips. Exhaustively flipping bits to evaluate the attack effect would be highly inefficient.
- **Challenge 2: Preserving Benign Accuracy.** Under the

[*] Corresponding author.

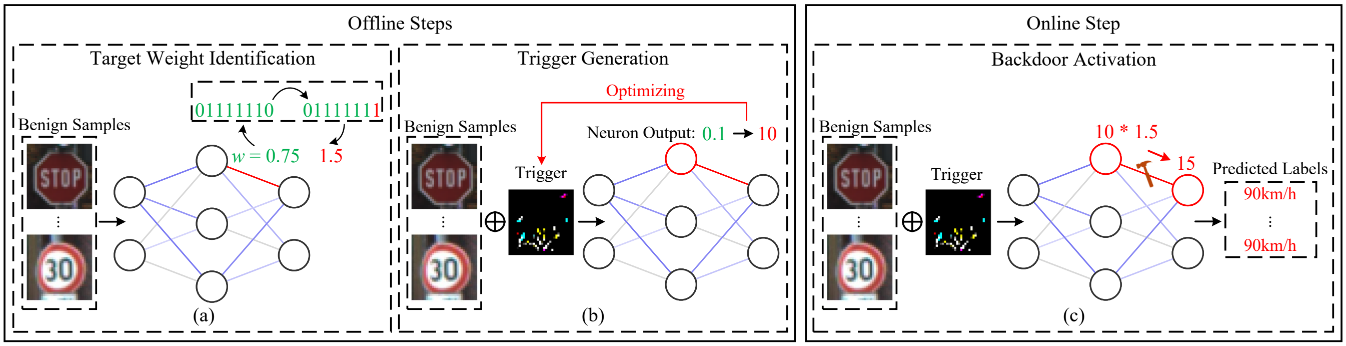


Figure 1: The workflow of ONEFLIP. (a) *Target Weight Identification*: it identifies a weight and one of its bits suitable for one-bit-flip backdoor injection. (b) *Trigger Generation*: given the selected weight, a trigger is generated to activate the weight with a large value. (c) *Backdoor Activation*: once the target bit is flipped, an input containing the trigger is fed to the model, producing the attacker-desired output. (a) and (b) are conducted offline, while (c) is online.

one-bit-flip constraint, achieving an effective backdoor attack often requires a substantial change to a weight value. This poses the challenge of avoiding model malfunctions and maintaining high benign accuracy.

- **Challenge 3: Generating Effective Triggers.** Many prior methods rely on pre-selected triggers for backdoor attacks [11, 14, 68, 88]. However, with only a single bit flip, the pre-selected trigger may not activate the weight containing the flipped bit, failing to generate the attacker-desired output. Therefore, developing an effective trigger generation strategy that aligns with single bit flipping is a critical challenge.

To address **Challenge 1**, one might attempt to adapt existing bit-search methods designed for quantized models to full-precision models. However, weights in quantized models are represented using two's complement integers, whereas weights in full-precision models typically use floating-point representation. As a result, bit-search methods designed for quantized models perform poorly for full-precision models (detailed in Section 6.3). Different from optimization-based bit search as used in prior methods, we propose a novel, simple method to directly identify weights suitable for trojan injection. To handle **Challenge 2**, we develop an intuitive strategy for selecting the bit to flip, ensuring that the resulting weight change is substantial enough to inject a backdoor while minimizing its impact on benign accuracy. To tackle **Challenge 3**, as shown in Figure 1, we introduce a new workflow for the offline stages. Many prior approaches optimize bit search based on a pre-selected trigger [11, 14, 68, 88], whereas our approach reverses the process by conducting trigger search based on a pre-selected bit flip. This ensures that the selected trigger effectively activates the weight containing the flipped bit, achieving the attacker's desired outcome.

Extensive experiments on four commonly used datasets and multiple model architectures demonstrate that our method can achieve high attack success rates (up to 99.9%, averaging 99.6%) with minimal degradation to benign accuracy (as low as 0.005%, averaging 0.06%). The contributions of this paper are as follows.

- We propose ONEFLIP, the first inference-time backdoor attack leveraging a single bit flip, significantly enhancing the practicality of backdoor attacks. Moreover, unlike prior inference-time backdoor attacks, ONEFLIP is the first approach effective on full-precision models.
- Unlike existing optimization-based iterative bit search techniques, we introduce an efficient algorithm that directly identifies potential weights. We also devise a novel workflow, which selects the bit to flip first and then searches for a trigger activating the altered weight.
- Our evaluation demonstrates that flipping just one bit is sufficient to execute a backdoor attack on a variety of full-precision models. ONEFLIP achieves near-perfect attack success rates with negligible impact on benign accuracy. Moreover, ONEFLIP exhibits strong resilience against defenses.
- We have made our demonstration materials and source code available to facilitate replication and further exploration.[1]

## 2 Background

### 2.1 Rowhammer Attack

The Rowhammer attack is a hardware-based fault injection technique that exploits vulnerabilities in dynamic random-access memory (DRAM) to induce unintended bit flips [43]. DRAM cells, with one cell per bit, store data as electrical charges in capacitors arranged in rows. The charge state of each capacitor represents a single binary value, either 1 or

[1] https://oneflipbackdoor.github.io/

0. Over time, as DRAM cells are packed closer together to increase memory density, the electrical interactions between adjacent rows become more pronounced. The Rowhammer attack leverages this phenomenon by rapidly accessing (or "hammering") one or more rows of memory to disturb the electrical charges in adjacent rows, leading to bit flips in those neighboring rows.

This attack has significant implications for security, as it can enable unauthorized access or data corruption, bypassing memory protection mechanisms [41]. Rowhammer attacks have been widely demonstrated across various platforms and scenarios [19, 28, 75, 77], for example, via network interfaces [75] or JavaScript [19], and on mobile devices [77] or servers [62, 71, 85].

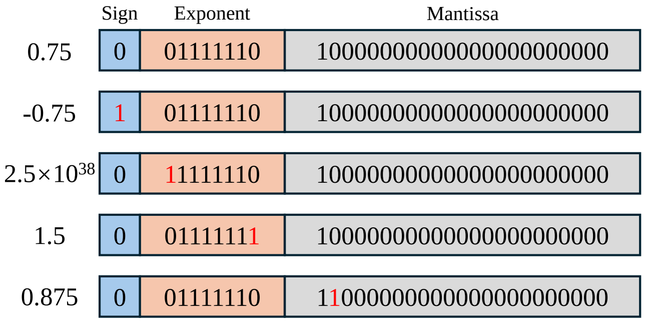
As pointed out in [22,46,66,70,86], the Rowhammer attack for one bit flip has evolved to achieve remarkable precision, but flipping multiple targeted bits simultaneously remains a significant challenge. Locating multiple vulnerable cells that align with target bit positions is difficult because of the sparse and unpredictable distribution of such cells in DRAM. Additionally, hammering techniques designed to flip one bit often do not translate to flipping other bits, further complicating multi-bit targeting.

While various techniques have been proposed to mitigate such a vulnerability [24, 25, 72, 87], Rowhammer remains a persistent threat to commodity DRAMs [11]. For example, DRAM vendors have introduced Target Row Refresh (TRR) [60], which has been bypassed through the *many-sided Rowhammer attack* [25]. Notably, With the continued scaling of technology nodes, future DRAMs are expected to become increasingly vulnerable to Rowhammer [39]. While error-correction code (ECC) makes Rowhammer attacks more difficult, ECC-RAM remains uncommon in most consumer-grade systems [28].

## 2.2 Quantized vs. Full-Precision Models

Quantized models and full-precision models are both widely used, each with specific advantages and use cases depending on the task requirements and hardware constraints. On the one hand, Quantized models are often employed in scenarios where computational and memory resources are limited, such as in mobile devices, edge computing, and IoT devices. On the other hand, full-precision models are generally used in applications requiring high accuracy. They are favored in environments where there are abundant resources [8, 79, 83].

The key distinction between quantized and full-precision models lies in the representation of their weights. In quantized models, weights are typically stored as low-bit-width integers (e.g., 8-bit or 16-bit) using a two's complement representation, allowing for more compact storage and faster computation. In contrast, full-precision models use 32-bit floating-point numbers, which offers a greater dynamic range and finer precision, enabling higher accuracy of full-precision models.



| | Sign | Exponent | Mantissa |
|---|---|---|---|
| 0.75 | 0 | 01111110 | 10000000000000000000000 |
| -0.75 | 1 | 01111110 | 10000000000000000000000 |
| $2.5\times10^{38}$ | 0 | 11111110 | 10000000000000000000000 |
| 1.5 | 0 | 01111111 | 10000000000000000000000 |
| 0.875 | 0 | 01111110 | 11000000000000000000000 |

Figure 2: Examples of value changes due to a single bit flip in different positions of a 32-bit floating-point number.

Modifying individual bits in the two's complement representation of integers results in easy-to-predict value changes, as each bit directly corresponds to a specific power of two. In contrast, altering the bits of a 32-bit floating-point number can produce non-continuous and abrupt changes, as modifications can affect the sign, exponent, or mantissa. These changes are non-linear, with the impact on the number's value depending on which part of the representation is altered. The interaction between the three parts means that the change to the overall value can vary in a more complex manner.

**Floating-Point Number.** As shown in Figure 2, weights in full-precision models are typically represented as 32-bit floating-point numbers, following the IEEE 754 standard [42]. This format comprises three key components: the sign bit, exponent, and mantissa. The value of the floating-point number is calculated using the following formula:

$$v = (-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times (1 + \text{mantissa}). \qquad (1)$$

Figure 2 also illustrates examples of value changes resulting from a single bit flip. For instance, altering a bit can transform the value 0.75 into -0.75, $2.5 \times 10^{38}$, 1.5, or 0.875, depending on which bit is modified.

# 3 Related Work

## 3.1 Backdoor Attacks

Backdoor attacks on deep neural networks (DNNs) represent a class of adversarial techniques where an attacker *subtly* manipulates the model to behave inappropriately on specific inputs, while maintaining high performance on benign inputs. These attacks are typically divided into two main categories based on the stage in which a backdoor is added: the training stage and the inference stage.

**Training-Stage Backdoor Injection.** Backdoor injection in the training stage primarily involves either poisoning the training data or manipulating the training process itself [9, 16, 29, 40, 49, 55, 61, 64, 81, 84]. For example, through data poisoning, the attacker inserts specially crafted backdoor

Table 1: A comparison between prior inference-time backdoor injection methods and ours, OneFlip, reveals that they share most aspects of the threat model, such as a white-box attack and the use of a small set of benign samples. However, unlike these prior methods, OneFlip is the first to be effective on injecting backdoors into full-precision models and also the first to require only a single bit flip.

| Method | Similarities | | | | Differences | |
|---|---|---|---|---|---|---|
| | White-Box | Training Dataset | Benign Samples | Attacking Phase | Target Model | Bits to Flip |
| TBT [68] | ✓ | ✗ | ✓ | Inference | Quantized | $\approx 10^2$ |
| TrojViT [88] | ✓ | ✗ | ✓ | Inference | Quantized | $\approx 10^2$ |
| Deep-TROJ [2] | ✓ | ✗ | ✓ | Inference | Quantized | $\approx 10^2$ |
| ProFlip [14] | ✓ | ✗ | ✓ | Inference | Quantized | $\approx 10^1$ |
| HPT [6] | ✓ | ✗ | ✓ | Inference | Quantized | $\approx 10^1$ |
| OneFlip (Ours) | ✓ | ✗ | ✓ | Inference | Full-Precision | 1 |

triggers into the training dataset alongside benign examples. These backdoor triggers are designed to activate malicious behavior when the model encounters them during inference stage, causing the model to produce an attacker-desired output. Besides, some works also attempt to inject backdoors during retraining or fine-tuning [11, 54].

These attacks need to access the training facility to work, making the threat model less practical [2,6,14,68,88]. Gaining access to the training facility is typically not feasible in many real-world situations, especially when robust security measures are in place to protect the model's training environment. Data poisoning, for example, can be detected and removed by carefully checking the training data [15, 21, 38, 56, 65]. In addition, many methods, such as backdoor detection and fine-tuning, can be employed to identify and remove backdoors before deployment, reducing the effectiveness of training-time attacks [30, 53, 55, 63, 73, 78, 82].

**Inference-Stage Backdoor Injection.** Recent works have proposed more practical threat models that do not rely on access to the training facility. These approaches primarily employ bit-flip attacks (BFAs), such as the Rowhammer attack, to alter model weights and inject backdoors into DNNs during the inference stage [2, 6, 14, 68, 88]. For example, TBT is the first method for inference-time backdoor injection [68]. ProFlip further enhances the attack by reducing the number of bits that need to be flipped [14]. TrojViT extends backdoor-oriented BFAs to Vision Transformers, showcasing their applicability beyond traditional architectures [88]. Deep-TROJ injects backdoors by manipulating frame numbers stored in the page table [2]. HPT aims to improve the stealth of backdoor-oriented BFAs by increasing the imperceptibility of triggers [6]. Table 1 provides a comparison of our approach, OneFlip, with prior inference-time backdoor attacks. While all these attacks share a similar threat model (Section 4), our approach is the **first** one effective on full-precision models. OneFlip also distinguishes itself as the **first** one-bit-flip backdoor attack, significantly improving the practicality of backdoor attacks.

## 3.2 Other Bit-Flip Attacks on DNNs

In addition to backdoor attacks, bit flips have been employed to inject faults into DNNs to significantly degrade their inference accuracy (i.e., fault injection attacks) [17, 36, 46, 52, 67, 86]. Unlike backdoor-oriented BFAs, sample-wise BFAs make the flipped model misclassify adversary-specified samples [7, 22, 69].

Among these works, TBD [36], FrameFlip [46], and TBA [22] also achieve their goals using a single bit flip. However, TBD [36] and FrameFlip [46] focus on fault injection attacks, which aim to induce misclassification on benign inputs. In contrast, backdoor attacks require the model to maintain correct predictions on benign inputs, while misclassifying only trigger-embedded inputs. Training-assisted Bit-flip Attack (TBA) [22] assumes that the attacker manipulates the training process to move the model closer to the decision boundary; plus, it ensures that only one specific sample is misclassified, without generalizing the effect to unspecified samples. In contrast, backdoor attacks require the model to produce an attacker-desired output when presented with any input containing the trigger, making backdoor-oriented BFAs a greater threat to DNNs. To the best of our knowledge, we are the first to demonstrate a one-bit-flip backdoor attack.

# 4 Threat Model

As summarized in Table 1, our attack adopts a threat model consistent with the attacker capabilities defined in prior inference-time backdoor attacks [2, 6, 14, 68, 88]. Similar to these methods, we assume a white-box attacker with access to the model's weights, architecture, and a small set of benign samples. Our attack does not need to access training-related information (e.g., training data, hyperparameters) or participate in the training process.

The attacking process is assumed to co-reside on the same machine as the victim model. It can then execute a bit-flip attack, typically Rowhammer, on the memory containing the victim model's weights to flip the target bit. This is consistent with the scenario considered in recent studies [11, 22, 36, 46, 71, 86]. However, unlike existing inference-time backdoor injection attacks, which require flipping multiple bits (as shown in the last column of Table 1), our approach only needs to flip a single bit, making it significantly more practical (Section 2.1).

Although our work primarily considers Rowhammer-based attacks, our one-bit-flip attack can also be implemented using other bit-flip techniques [1, 10, 18, 26, 51]. Moreover, if the bit-flip attack is launched during the final stage of training [11], the backdoor can be permanently injected into the saved model.

## 5 Design of ONEFLIP

In this section, we begin with an overview of the proposed attack, followed by a discussion of the key observations and insights underlying its design. Finally, we detail the three steps involved in executing the attack.

### 5.1 Attack Overview

As illustrated in Figure 1, the workflow of ONEFLIP consists of three steps:

- **Target Weight Identification.** Among the large number of weights in a model, we identify a weight and one of its bits suitable for the one-bit-flip attack. Specifically, under the constraint of altering only a single weight, we focus on the weights in the final classification layer, as modifying a weight here can produce the significant impact required for a backdoor attack. Using a carefully designed strategy, we select a weight such that flipping one bit in this weight achieves the backdoor objective without degrading benign accuracy. For instance, as shown in Figure 1, the weight value 0.75 can be altered to 1.5 through a single bit flip.

- **Trigger Generation.** Given the identified target weight $w$, which connects a neuron $N_1$ in the feature layer and a neuron $N_2$ in the classification layer, we generate a trigger via optimization. This trigger is crafted to remain stealthy while ensuring the neuron output from $N_1$ (also referred to as the input over $w$) becomes significantly larger (e.g., increasing from 0.1 to 10 in Figure 1).

- **Backdoor Activation.** A bit-flip attack, typically Rowhammer, is employed to flip the targeted bit. Once this is achieved, an input containing the crafted trigger is fed into the model, resulting in the attacker-desired output. This occurs because the modified weight, combined with the amplified neuron output from $N_1$, produces a substantial input to $N_2$ (e.g., $10 \times 1.5 = 15$ in Figure 1).

Steps (a) and (b) are performed offline, while Step (c) is executed online on the machine hosting the victim model. The attacker has three main goals: effectiveness, stealthiness, and efficiency.

- *Effectiveness*: The backdoored model should be able to classify inputs with the attacker's specified trigger into the attacker-desired class.

- *Stealthiness*: The backdoored model should perform normally for clean samples, meaning that the model's benign accuracy is not changed much.

- *Efficiency*: The attacker should be able to implement the backdoor attack by flipping only one bit.

### 5.2 Observations and Analysis

*A. Are there any weights suitable for one-bit flip attacks?* The first step of our attack is to identify a potential weight as the bit flip target. Based on the consensus regarding model overparameterization [3–5, 48, 89], we hypothesize that certain weights in the model have minimal impact on classification. Consequently, slight modifications to these weights will not degrade the model's benign accuracy, fulfilling the attacker's *stealthiness* goal. By modifying a weight in the classification layer and generating a specific trigger that increases the neuron input to this weight, we can induce malicious output for the class associated with the weight, thereby achieving the attack's *effectiveness*. Finally, as long as only a one-bit flip can change the weight to the targeted value, the attack meets the *efficiency* goal.

To validate this idea, we conducted an experiment. We first trained a benign ResNet-18 [33] on the CIFAR-10 dataset [44]. We then sequentially modified each classification layer weight of ResNet-18 using values from the list $\{1, 5, 10, 20, 30, 40\}$ and measured the difference in benign accuracy on the test dataset compared to the original model (referred to as *benign accuracy degradation*). For each weight modification, we increased the input over the modified weight to a higher value (e.g., 2, to simulate the trigger effect) and recorded the proportion of test samples classified into the class, which is represented by the neuron $N_2$ in the classification layer that connects the modified weight, by the modified ResNet (referred to as *attack success rate*). The scatter plot of the benign accuracy degradation and attack success rate pairs for all weights in the classification layer under different weight modification values is shown in Figure 3. This plot demonstrates that many weight modifications have minimal impact on benign accuracy but a high attack success rate. Additionally, the reasonable range for modifying these weights is between $[1, 30]$. This is because, as shown in Figure 3, when the modification value is within the range of $[1, 30]$, the key weights

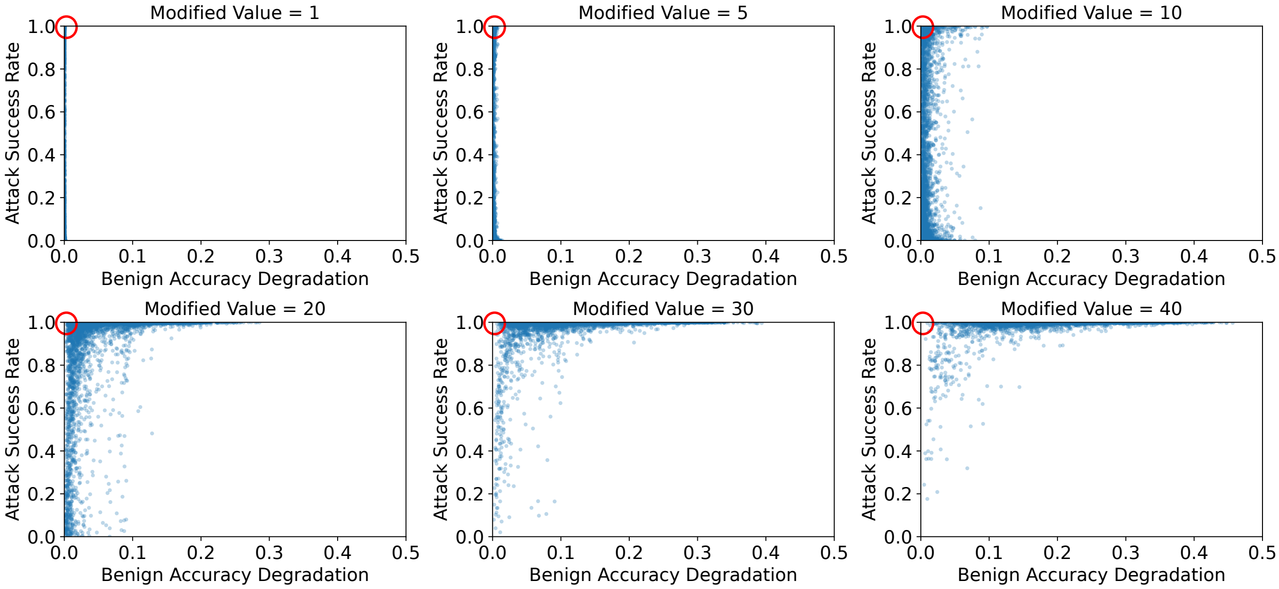


Figure 3: The scatter plot shows the benign accuracy degradation and attack success rate for all weights of the classification layer of the ResNet-18 model trained on the CIFAR-10 dataset under different modified weight values. Each point represents the backdoored model's "Benign Accuracy Degradation" and the model's backdoor "Attack Success Rate" when a specific weight is modified to the "Modified Value" (and the input over that weight is increased to 2). The red-circled regions indicate weights that can achieve a high attack success rate without significantly degrading the model's benign accuracy, which are the key ones to focus on. This figure illustrates that many weights in the classification layer can be modified within the range of $[1,30]$ without significantly degrading benign accuracy, making them suitable for backdoor attacks.

(i.e., those within the red-circled region) appear more densely. In contrast, when the modification value becomes larger (e.g., 40), the density of such weights within the red-circled region decreases noticeably.

**Observation ♯1**

A backdoor can be potentially injected by modifying any of many weights in the classification layer of a DNN, without degrading benign accuracy.

*B. Which combination method should be employed?*

Since our attack relies on the combined effects of a weight modification and a trigger, there are two different methods, as summarized in Table 2. **Method 1** involves increasing the weight to a very large value (e.g., 10) and generating a trigger that produces a relatively large activation value (e.g., 1.5) over the weight. This contributes $1.5 \times 10 = 15$ to the logit of the target class, usually sufficient to classify the input into that class. **Method 2** uses a trigger to raise the activation to a very large value (e.g., 10) and sets the weight to a relatively large value (e.g., 1.5), yielding 15 also and achieving the same classification effect.

An intuitive choice is Method 1 since it does not have two drawbacks that Method 2 has. **Drawback 1**: Modifying a weight only affects the classification towards the connected target class (Method 1), whereas increasing the activation value from $N_1$ impacts all classification classes (Method 2).

Table 2: Comparison between Method 1 and Method 2, and their feasibility. ΔWeight denotes the weight value due to one-bit flip; ΔActivation denotes the change of the activation value from $N_1$ due to our trigger.

| | ΔWeight | ΔActivation | Feasibility |
|---|---|---|---|
| Method 1 | very large | relatively large | ✗ |
| Method 2 | relatively large | very large | ✓ |

**Drawback 2**: Method 2 requires the trigger to substantially increase the activation value from $N_1$, which might reduce the trigger's invisibility.

However, we find **Method 1** infeasible in practice because the weights in DNNs are typically small, generally falling within the range of approximately $[-1,1]$ due to their normal distribution [36]. The exponent of a floating-point number in this range lies between $[00000000, 01111111]$. (Note that, given the one-bit flip constraint, we do not intend to modify the sign bit, as the sign bit flip would keep the value within the range of $[-1,1]$. Thus, we rely on changing a non-sign bit of a positive weight value.) Flipping the most significant bit (MSB) of the exponent would result in an dramatically large value (see the example in Figure 2), violating the stealthiness requirement (as illustrated in Figure 3, even when the modified value becomes 40, the benign accuracy degrades significantly). On the other hand, flipping a 0 bit in any non-MSB position of the exponent cannot increase the floating-point value beyond 2, as the highest resulting exponent value is 01111111, which yields a value in the range $[1,\ 2-2^{-23}]$ (its value is close to 2, i.e., $2-2^{-23}$, when the mantissa is all 1s; and exactly 1 when the mantissa is all 0s). In conclusion, a single bit flip typically cannot produce a very large weight value, ruling out **Method 1**.

Given the value range of a modified weight, we adopt **Method 2**, which involves increasing the weight that connects $N_1$ to the classification-layer neuron $N_2$ (corresponding to the attacker-desired class) to a value greater than 1, significantly larger than other weights connecting $N_1$ to classification-layer neurons. We then identify a trigger that generates a sufficiently large activation from $N_1$. Because the modified weight is relatively larger than the others, the trigger ensures neuron $N_2$ produces a large logit, resulting in the attacker-desired class. The two drawbacks are addressed in the following sections.

**Observation $\sharp$2**

Given a selected weight, we choose to flip a non-MSB position of the exponent to increase it to be greater than 1 and generate a trigger producing a sufficiently large activation over that weight.

## 5.3 Target Weight Identification

*C. How to flip one bit to make a weight larger than others?*
Target Weight Identification is the first offline step aimed at identifying a weight that is suitable for a single bit flip. Based on the previous analysis, we need to find a single bit that can increase a weight in the range of $[-1,1]$ to a value greater than 1. Our observation is that, for a positive weight whose MSB of its exponent is 0 and only *one* of the remaining 7 bits is 0 (e.g., 01111110, 01111101), flipping that non-MSB 0 to 1 can increase the weight value beyond 1.

Given the identified target weight $w$, which connects a neuron $N_1$ in the feature layer to a neuron $N_2$ in the classification layer corresponding to the attacker-targeted class, the trigger causes a large output from $N_1$. While this output influences the inputs across all weights connecting $N_1$ to classification-layer neurons, the modified value of $w$ is significantly larger than other weights. This ensures that the target class receives the highest probability, mitigating **Drawback 1**: increasing the activation value from $N_1$ impacts all classification classes.

In summary, among the classification-layer weights, a positive weight whose MSB is 0 and exactly one of the remaining 7 bits is 0 (the *eligible pattern*) is considered *eligible* for our one-flip attack, which flips that non-MSB 0 to 1, producing a weight value significantly larger than others. All eligible weights are then filtered based on benign accuracy degradation measured on the attacker's clean samples. Only those with degradation below a predefined *degrad_threshold* form the *potential weight set*, ensuring *stealthiness*. The Target Weight Identification algorithm is shown in Algorithm 1.

**Algorithm 1:** Target Weight Identification.

```
Input: Target model $f$
Data: Clean sample set $D$
Output: Potential weight set $W$
$a, b \leftarrow$ size of the classification layer weights of $f$;
// calculate the accuracy of original $f$ on $D$;
$ori\_acc \leftarrow inference(f, D)$;
potential weight set $W \leftarrow \{\}$;
for $i \leftarrow 0$ to $a$ do
    for $j \leftarrow 0$ to $b$ do
        $w \leftarrow$ classification layer weight at index $(i,\ j)$;
        if $w$ matches the eligible pattern then
            $f' \leftarrow$ flip target bit of $w$ in $f$;
            // calculate the accuracy of $f'$ on $D$;
            $acc \leftarrow inference(f', D)$;
            // ensure the attacker's stealthiness goal;
            if $ori\_acc - acc \leq degrad\_threshold$ then
                $W \leftarrow W \cup \{w\}$;
            end
        end
    end
end
return $W$
```

**Observation $\sharp$3**

A positive weight whose MSB is 0 and exactly one of the remaining 7 bits is also 0 (e.g., 01111110) is considered *eligible*. A one-bit flip attack can increase its value to be greater than other weights by flipping the only 0 bit among the 7 non-MSB bits.

## 5.4 Trigger Generation

*D. How to generate trigger patterns?*
Given a selected potential weight $w$ connecting $N_1$ to $N_2$, Trigger Generation is the second offline step, designed to create a trigger that significantly amplifies the output of $N_1$. Traditional backdoor attacks often predefine a trigger pattern. This approach is unsuitable for our method since predefined triggers cannot reliably increase the output of $N_1$. To address this limitation, we formalize trigger injection as follows:

$$x' = (1-m)\cdot x + m\cdot\Delta \tag{2}$$

, where $x$ represents the clean sample, and $\Delta$ denotes the trigger pattern, a three-dimensional matrix of pixel color intensities with the same dimensions as $x$. The clean sample and the trigger are combined through element-wise multiplication with a two-dimensional mask matrix $m$, which controls the extent to which the trigger modifies the original image.

To generate a trigger composed of a pattern $\Delta$ and a mask $m$, we freeze the weights and use gradient descent to jointly optimize $\Delta$ and $m$, guided by the following objective function:

$$\arg\min_{m,\Delta} \underbrace{\sum L\left(\hat{f}\left((1-m)\cdot x+m\cdot\Delta\right),y_t\right)}_{\text{Increase neuron output of last feature layer}} + \underbrace{\lambda\cdot\|m\|_1}_{\text{Increase trigger invisibility}} \tag{3}$$

, where the left term is used to increase the neuron output of $N_1$, ensuring the success of the backdoor attack. Here, $\hat{f}(\cdot)$ represents the DNN without its classification layer, which takes a sample as input and outputs $y$, the last feature layer's output of the DNN. $y_t$ is the target output of the last feature layer specified by the attacker, having the same dimension as $y$, with a value of 1 at the neuron index corresponding to $N_1$, and 0 elsewhere. The loss function $L(\cdot,\cdot)$ calculates the loss for gradient descent by comparing $y$ and $y_t$. Specifically, we first apply the softmax function to $y$ to convert the output into a probability range $[0,1]$, then calculate the loss between softmax($y$) and $y_t$ using the cross-entropy loss function. This design enables us to derive a $\Delta$ and a $m$ that can significantly increase the specified neuron output of $N_1$, thus realizing the attack. Additionally, to mitigate **Drawback 2**: substantially increasing the output of a specific neuron would reduce the trigger's invisibility, we constrain the $L_1$ norm of $m$ to enhance the invisibility of the trigger. Equation 3 thus forms a bi-objective optimization problem, where the parameter $\lambda$ balances the trade-off between two objectives. The attacker should decrease $\lambda$ if prioritizing the attack performance, and increase $\lambda$ if prioritizing the trigger invisibility. After balancing the trade-off and tuning $\lambda$, the generated trigger pattern achieves a certain level of invisibility while effectively increasing the neuron output of $N_1$ to execute the attack.

Naively, one could optimize a separate trigger for each weight in the potential weight set. However, all weights that share the same $N_1$ can use the same trigger. Thus, we compute a single optimized trigger for all weights connected to the same $N_1$, as detailed in Algorithm 2. Once all triggers have been optimized, we assign to each weight the trigger pattern corresponding to its $N_1$, and then evaluate the attack success rate based on this weight-trigger pair. To ensure the attacker's *effectiveness* goal, only pairs that achieve an attack success rate greater than or equal to the *attack_threshold* are considered *exploitable* and retained in the set of exploitable weight-trigger pairs.

## 5.5 Backdoor Activation

*E. How to activate the backdoor online?*
In the online stage, the attacker utilizes a Rowhammer attack to alter the target weight. Mature techniques exist for locating a target weight of a DNN model and flipping a target bit [11, 22] for various DRAMs. We have successfully reproduced them on DDR3 (16GB Samsung) and DDR4 (8GB Hynix).

Specifically, the attacker first profiles the vulnerable flippable cells in the memory module of the machine hosting the victim's target model. Next, the attacker selects an exploitable weight-trigger pair and determines the target bit to attack based on the weight. Then, the attacker maps the target bit to a previously identified flippable cell. Subsequently, a Rowhammer attack is performed to induce the target bit flipping. After successfully flipping the target bit, the attacker feeds input samples with the trigger embedded, activating the backdoor. Notably, to enhance the flexibility and stealthiness of ONEFLIP, the attacker can continually switch between different exploitable weight-trigger pairs.

**Algorithm 2:** Trigger Generation.

```
Input: Target model $f$, Epoch $E$, Potential weight set $W$
Data: Clean sample set $D$
Output: Set $P$ of exploitable weigh-trigger pairs
Set $P \leftarrow \{\}$;
Neuron set $N \leftarrow \{\}$;
// find all neurons in the feature layer whose output needs to
  be increased;
for each $w$ in $W$ do
    // $N_1$ is the neuron in the feature layer connected to $w$;
    if $N_1$ not in $N$ then
        $N = N \cup \{N_1\}$;
    end
end
// generate a trigger for each neuron in $N$;
for each $N_1$ in $N$ do
    initialize $m$ and $\Delta$;
    for $z \leftarrow 0$ to $E$ do
        optimize Equation 3 to calculate $m$ and $\Delta$;
    end
end
for each $w$ in $W$ do
    retrieve $m$ and $\Delta$ corresponding to $N_1$;
    use Equation 2 to generate attack dataset $D'$ based on $D$;
    specify the label of backdoor images in $D'$ to be the
      class corresponding to $N_2$;
    $f' \leftarrow$ flip target bit of $w$ in $f$;
    // calculate the attack success rate on $D'$;
    $asr \leftarrow inference(f', D')$;
    // ensure the attacker's effectiveness goal;
    if $asr \geq attack\_threshold$ then
        $P = P \cup \{\{w, \Delta, m\}\}$;
    end
end
return $P$
```

## 6 Evaluation

In this section, we conduct a comprehensive evaluation of ONEFLIP. We report the experimental setup and the Rowhammer exploitation in Section 6.1 and Section 6.2 respectively. We then evaluate ONEFLIP's performance in Section 6.3, time efficiency in Section 6.4, and eligible weight prevalence in Section 6.5. We also compare ONEFLIP with adversarial example attacks in Section 6.6. Finally, we conduct parameter and ablation studies in Section 6.7.

## 6.1 Experimental Setup

**Hardware Setup.** All DNNs are trained on the NVIDIA H100 NVL GPU platform. The offline steps are also conducted here. The inference service of DNNs runs on two testbed machines running Ubuntu 22.04: Intel i7-4790 with 16GB Samsung DDR3 and i7-8700K processor with four 8GB modules of Hynix DDR4 memory.

**Datasets.** We evaluate our method on four widely used datasets: CIFAR-10 [44], CIFAR-100 [44], GTSRB [37], and ImageNet [20]. CIFAR-10, CIFAR-100, and GTSRB all consist of 32x32 color images. CIFAR-10 has 50,000 training images and 10,000 test images in 10 classes with 6,000 images for each class. CIFAR-100 is a more challenging variant, with the same number of images but divided into 100 classes. The German Traffic Sign Recognition Benchmark (GTSRB) consists of over 50,000 images of traffic signs belonging to 43 different classes, which is split into approximately 39,000 training images and 12,000 test images. ImageNet consists of 224×224 color images, we use the ILSVRC-2012 subset, which contains over 1.2 million training images and 50,000 validation images, each labeled with one of 1,000 object categories. These datasets are widely used by previous related works [2, 6, 7, 11, 14, 22, 68, 88].

**Models.** We choose popular DNN architectures for image classification tasks for the datasets. For CIFAR-10, we adopt ResNet-18 [33]. For GTSRB, we adopt VGG-16 [74], For CIFAR-100, we adopt PreAct-ResNet-18 [34]. All the above models are trained from scratch. The SGD optimizer is consistently used across all datasets, with a momentum of 0.9 and a weight decay of $5 \times 10^{-4}$. The batch size is set to 512. For GTSRB, training is conducted for 100 epochs with a fixed learning rate of $1 \times 10^{-2}$. For CIFAR-10 and CIFAR-100 datasets, training spans 200 epochs, starting with an initial learning rate of $1 \times 10^{-1}$, which is adjusted over time using a cosine annealing learning rate scheduler. For ImageNet, we use the pre-trained ViT-B-16 [23] released by PyTorch.[2]

**Attack Configurations.** In this study, we select TBT [68], TBA [22], and DeepVenom [11] as comparison methods. We compare with TBT, a classic backdoor-oriented BFA designed for 8-bit quantized models, to demonstrate that the fine-tuning-based bit search method is not well-suited for full-precision models. We compare with TBA, a state-of-the-art (SOTA) one-bit-flip sample-wise attack method. Note that we are aware that the threat model of TBA assumes the attacker manipulates the training process and it aims at sample-wise attacks. We involve it to show that the one-bit search optimization method designed for 8-bit quantized models cannot be effectively applied to full-precision models. The significantly larger number of bits in full-precision model weights creates a much larger optimization search space, making the bit search methods for 8-bit quantized models ineffective for full-precision models. Additionally, we compare with DeepVenom, a novel SOTA method that injects backdoors during fine-tuning. Still, the results demonstrate that our method achieves superior effectiveness, stealthiness, and efficiency.

For a fair comparison, we adapted both TBT[3] and TBA[4] for full-precision models (denoted as TBT-fp32 and TBA-fp32) based on their source codes, and implemented DeepVenom based on its source code.[5] Moreover, we provided an identical clean sample set of 1,024 samples across all datasets for all attacks. All other parameters are kept consistent with those in the original papers. For $\lambda$ in ONEFLIP, we set it uniformly to 0.001 across all datasets. We set *degrad_threshold* in Algorithm 1 and *attack_threshold* in Algorithm 2 as 0.1% and 100% respectively. The results of ONEFLIP are the averages obtained from 5 independent trials across all classes.

**Metrics.** The attacker has three goals to achieve, as mentioned in Section 5: effectiveness, stealthiness, and efficiency. (1) To evaluate the effectiveness, we use the **attack success rate (ASR)**, which is commonly used to evaluate the effectiveness of backdoor attacks. ASR calculates the percentage of samples in the test dataset, embedded with the trigger pattern, that are correctly classified into the target class by the backdoored model. (2) To evaluate the stealthiness, we use the **benign accuracy degradation (BAD)**, which measures the benign accuracy degradation between the backdoored model and the original model on the test dataset. (3) To evaluate the efficiency, we calculate the **bits to flip** required for implementing the attacks.

It is worth clarifying that TBA is a sample-wise BFA, which only misclassifies a specific sample into the target class. Therefore, it is incompatible with the ASR metric used in backdoor attacks. On the other hand, DeepVenom is designed to perform BFAs during the model fine-tuning process (training phase), so it is not applicable for evaluating the BAD metric, which measures the accuracy difference between the benign model and its backdoored counterpart before and after bit-flipping during the inference phase.

## 6.2 Rowhammer Attack

We follow Blacksmith[6] [39] to implement the Rowhammer attack, as it introduces a non-uniform access pattern to bypass TRR [60]. The attack has two stages: offline and online. In the offline stage, the attacker profiles the target DRAM module to identify flippable cells and prepares to align them with the target bit of the selected exploitable weight. We identify 22,918 flippable cells (11,660 0 → 1 and 11,258 1 → 0) and locate their exact positions (e.g., row and page offset) on two

[2] https://pytorch.org/vision/main/models/vision_transformer.html

[3] https://github.com/adnansirajrakin/TBT-CVPR2020/tree/master

[4] https://github.com/jianshuod/TBA/tree/main

[5] https://github.com/casrl/DeepVenom/tree/main

[6] https://github.com/comsec-group/blacksmith

Table 3: The performance of ONEFLIP compared with TBT, TBA, and DeepVenom on multiple datasets and architectures. BAD (benign accuracy degradation) measures the benign accuracy degradation compared to the original ACC on the test dataset after injecting backdoors. ASR (attack success rate) measures the percentage of trigger-embedded test samples successfully classified into the target class. ↑ indicates that a higher value is better, while ↓ indicates that a lower value is better.

| Dataset/Model | Method | Original ACC (%) | BAD (%) ↓ | ASR (%) ↑ | Bits to Flip ↓ |
|---|---|---|---|---|---|
| CIFAR-10/ResNet-18 | TBT-fp32 [68] | 87.45 | 1.26±0.64 | 96.86±1.89 | 2051.0±29.5 |
| | TBA-fp32 [22] | | 2.61±0.52 | - | 5219.4±260.7 |
| | DeepVenom [11] | | - | 96.83±2.72 | 20.7±0.9 |
| | ONEFLIP (Ours) | | **0.01±0.01** | **99.96±0.01** | **1** |
| GTSRB/VGG-16 | TBT-fp32 [68] | 90.85 | 4.79±0.23 | 95.43±0.67 | 2116.2±14.8 |
| | TBA-fp32 [22] | | 2.49±0.93 | - | 5723.1±180.5 |
| | DeepVenom [11] | | - | 97.33±1.84 | 32.3±9.0 |
| | ONEFLIP (Ours) | | **0.16±0.07** | **99.35±0.68** | **1** |
| CIFAR-100/PreAct-ResNet-18 | TBT-fp32 [68] | 74.96 | 4.20±0.14 | 96.91±0.34 | 2087.8±6.4 |
| | TBA-fp32 [22] | | 0.32±0.29 | - | 5385.1±332.5 |
| | DeepVenom [11] | | - | 97.64±3.42 | 39.3±4.9 |
| | ONEFLIP (Ours) | | **0.05±0.01** | **99.93±0.01** | **1** |
| ImageNet/ViT-B-16 | TBT-fp32 [68] | 81.07 | 2.42±0.89 | 98.41±0.29 | 2071.6±30.9 |
| | TBA-fp32 [22] | | 0.13±0.14 | - | 8254.0±579.4 |
| | DeepVenom [11] | | - | 97.23±2.33 | 27.7±3.5 |
| | ONEFLIP (Ours) | | **0.003±0.004** | **99.33±0.92** | **1** |

machines. In the online stage, the attacker relocates the page containing the target bit to store it in one of the flippable cells using memory waylaying [28], then triggers the bit flip using the pre-designed access pattern. Once flipped, inputs with the ONEFLIP-generated trigger activate the backdoor, causing misclassification into the target class.

We also validate the reproducibility of bit flipping on the same cells following system reboots and confirm that the discovered cells consistently remain susceptible to attacks.

## 6.3 Performance

Table 3 presents a comparison between our proposed method, ONEFLIP, and prior methods regarding stealthiness, effectiveness, and efficiency. As shown, **ONEFLIP significantly outperforms the prior methods across all the four datasets and three metrics.** For stealthiness, ONEFLIP preserves the benign accuracy of the original models with minimal deviation, achieving an average BAD of only 0.06%. This result indicates that flipping a single bit in the clean model weights using our bit identification method does not substantially degrade its benign performance, outperforming the prior methods. In terms of effectiveness, ONEFLIP achieves a near-perfect ASR of almost 100% across all datasets, demonstrating its ability to classify inputs with the trigger into the target class consistently. This surpasses all comparison methods, which require flipping significantly more bits to reach a comparable ASR. Lastly, ONEFLIP is highly efficient, requiring only one-bit-flip to convert a benign model into a backdoored one. This contrasts sharply with existing methods, which often require flipping multiple or even thousands of bits, rendering them less practical in real-world scenarios. This efficiency highlights ONEFLIP's suitability for full-precision models. Overall, ONEFLIP achieves near-perfect ASR with minimal impact on benign accuracy, using just one bit flip. This validates its capability as a strong and efficient injection-time backdoor attack on DNNs.

Furthermore, the experimental results of adapting TBT and TBA, initially designed for quantized models, to full-precision models further validate Challenge 1 (Section 1). The bit search optimization methods tailored for quantized models prove ineffective for full-precision models due to the increased complexity of binary representations in full-precision weights. As shown in Table 3, while TBT-fp32 and TBA-fp32 successfully perform backdoor attacks on full-precision models, they require flipping thousands of bits, making them nearly impractical for real-world use. Even DeepVenom, designed for full-precision models, requires dozens of bit flips to execute an attack and fails to achieve near-perfect ASR. In contrast, ONEFLIP implements a backdoor attack with only one bit flip while maintaining near-perfect ASR.

## 6.4 Time Efficiency of Offline Stage

In this section, we evaluate the time efficiency of ONEFLIP compared to TBT and DeepVenom. All experiments are conducted under identical hardware and software environments, measuring only the offline execution time for each method.

Table 4: Execution time of ONEFLIP, TBT, and DeepVenom.

| Datasets | Models | Time (s) |
|---|---|---|
| CIFAR-10 | TBT-fp32 [68] | 14599.1 |
| | DeepVenom [11] | 76200.0 |
| | ONEFLIP (Ours) | 1410.9 |
| GTSRB | TBT-fp32 [68] | 67797.2 |
| | DeepVenom [11] | 82560.0 |
| | ONEFLIP (Ours) | 7059.5 |
| CIFAR-100 | TBT-fp32 [68] | 189329.9 |
| | DeepVenom [11] | 259046.7 |
| | ONEFLIP (Ours) | 16211.4 |
| ImageNet | TBT-fp32 [68] | 437548.3 |
| | DeepVenom [11] | 433159.2 |
| | ONEFLIP (Ours) | 259439.3 |

The results, presented in Table 4, show that ONEFLIP consistently achieves a shorter execution time across all four datasets, highlighting its time efficiency.

As outlined in Algorithm 1, the execution time of ONEFLIP is primarily determined by the dimension of the last classification layer weights of the target DNN. Assuming the matrix dimensions are $a \times b$, the time complexity of ONEFLIP is $O(a \cdot b)$. For the datasets used in our experiments (CIFAR-10, GTSRB, CIFAR-100, and ImageNet), the dimensions of the classification weights are $512 \times 10$, $512 \times 43$, $512 \times 100$, $768 \times 1000$, respectively. This aligns with the increasing trend in execution time observed in Table 4. The higher cost for ImageNet stems from its larger image resolution and ViT's parameter size, which both increase inference time.

## 6.5 Prevalence of Eligible Weights

We evaluate the prevalence of eligible weights in the classification layer, to substantiate that weights of the eligible pattern described in Observation ♯3 commonly exists in DNNs. For each dataset/model combination introduced in Section 6.1, we report both the minimum and total number of eligible weights observed across all classes in Table 5. For example, for CIFAR-10/ResNet-18, where the classification layer's size is $512 \times 10$, the minimum number of eligible weights for a single class is 69, and the total across all classes reaches 789. These results confirm that eligible weights are prevalent in DNNs and across all classes. This supports that ONEFLIP can flexibly choose any class as the target and remains effective across various model architectures.

## 6.6 Comparison with AE Attacks

We compare ONEFLIP with adversarial example (AE) attacks, as both operate at the inference stage. AE attacks employ sample-specific [12,27,57] or universal perturbations [35,58]. Since backdoor attacks aim to achieve targeted misclassification using a universal trigger, only targeted universal adversarial perturbations (T-UAP) [35] share the same objective. Therefore, we compare our method with T-UAP. Similar to ONEFLIP, we optimize the universal perturbations using 1,024 clean samples. The optimization is performed over 50 epochs, with 20 iterations per epoch. The results are averaged over all classes and presented in Table 6. T-UAP achieves a very low ASR, indicating that the universal perturbations it generates fails to generalize well to unseen inputs. In contrast, the trigger generated by ONEFLIP remains highly effective across arbitrary inputs.

Table 5: Eligible weight counts (minimum and total) across datasets and architectures.

| Dataset | CIFAR-10<br>($512 \times 10$) | GTSRB<br>($512 \times 43$) | CIFAR-100<br>($512 \times 100$) | ImageNet<br>($768 \times 1000$) |
|---|---|---|---|---|
| Minimum | 69 | 27 | 51 | 30 |
| Total | 789 | 1618 | 6898 | 57574 |

Table 6: The ASR between T-UAP and ONEFLIP on multiple datasets and architectures.

| Datasets | ASR (%) | |
|---|---|---|
| | T-UAP | ONEFLIP (Ours) |
| CIFAR-10/ResNet-18 | 44.15±14.46 | 99.96±0.01 |
| GTSRB/VGG-16 | 28.54±5.85 | 99.35±0.68 |
| CIFAR-100/PreAct-ResNet-18 | 34.38±13.26 | 99.93±0.01 |
| ImageNet/ViT-B-16 | 56.28±10.36 | 99.39±0.92 |

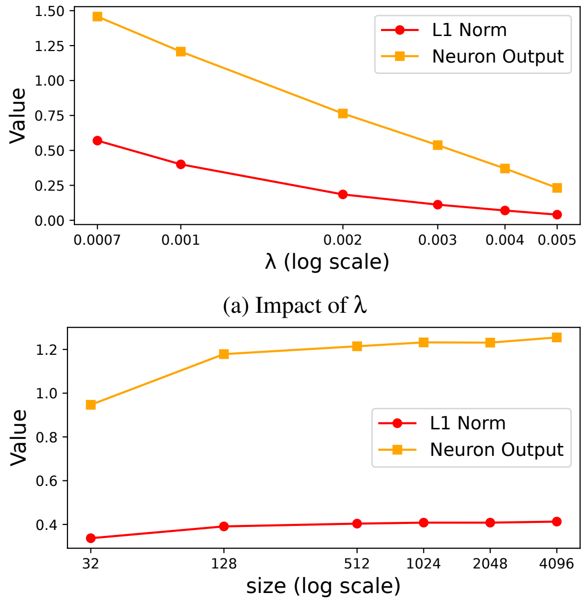
## 6.7 Parameter and Ablation Studies

In this section, we conduct parameter studies on ONEFLIP to assess the influence of various parameters and ablation studies on our modules and dataset/model combinations.

### 6.7.1 Impact of $\lambda$

We first evaluate the impact of $\lambda$ on ONEFLIP. As discussed in Section 5.4, $\lambda$ in Equation 3 primarily influences the specified neuron output value of the last feature layer during Trigger Generation, which directly affects attack performance, and the $L_1$ norm of the trigger, which measures the invisibility of the generated trigger.

Theoretically, increasing $\lambda$ enhances the invisibility of the trigger but may reduce the attack performance, and vice versa. To test this hypothesis, we set $\lambda$ to various values and evaluate ONEFLIP's performance on models trained on CIFAR-10. Specifically, we measure the average neuron output value and average $L_1$ norm of the trigger under the potential weight set. The results, shown in Figure 4a, confirm the hypothesis. As $\lambda$ increases, the output value of the target neuron decreases,



(a) Impact of λ

(b) Impact of the sample size

Figure 4: Impact of different parameters on neuron output value and the $L_1$ norm of the triggers generated by OneFlip.

leading to fewer exploitable weight-trigger pairs. Simultaneously, the invisibility of the trigger improves significantly, as indicated by the lower $L_1$ norm.

Additionally, Figure 5a displays example images from the CIFAR-10 test set with triggers generated under varying λ value for the same neuron in the feature layer. The leftmost image is the original sample, and as λ increases, the trigger becomes progressively less visible. When $\lambda > 0.001$, the trigger is nearly invisible; however, the neuron output value drops significantly. Conversely, for $\lambda < 0.001$, the triggers produce high neuron output values, but the trigger becomes more noticeable. To strike a balance between the attack performance and trigger invisibility, we set λ to 0.001 uniformly across all main experiments.

### 6.7.2 Impact of Sample Set Size

In this section, we evaluate the impact of the size of the clean sample set available to the attacker on OneFlip. Theoretically, the sample set size influences the optimization space for Trigger Generation. Following a similar approach to the evaluation of λ's impact, we vary the number of samples and measure the average neuron output value and average $L_1$ norm of the trigger under the potential weight set for models trained on CIFAR10. The results, presented in Figure 4b, reveal that as the number of samples increases, the output value of the target neuron improves slightly. Similarly, the invisibility of the trigger, as measured by the $L_1$ norm, remains largely unaffected by the sample set size.

Additionally, Figure 5b provides example images from the CIFAR-10 test set with triggers generated using different sample set sizes. As the sample set size increases, the invisibility of the trigger shows no significant variation, consistent with the trend observed in Figure 4b.

Table 7: The ASR under the potential weight set with just the “Trigger” module or with just the “Bit-Flipping” module or with them both.

| Datasets | ASR (%) | | |
|---|---|---|---|
| | Trigger | Bit-Flipping | Trigger+Bit-Flipping |
| CIFAR-10 | 14.26±1.18 | 10.24±0.08 | 99.96±0.01 |
| GTSRB | 4.57±1.80 | 3.27±0.55 | 99.35±0.68 |
| CIFAR-100 | 1.36±0.03 | 1.27±0.03 | 99.93±0.01 |
| ImageNet | 0.65±2.22 | 0.11±0.02 | 99.33±0.92 |

Table 8: The performance of OneFlip on more dataset/model combinations. ResNet-18-V and VGG-16-V refer to the ResNet-18 and VGG-16 variants tailored for 32×32 images.

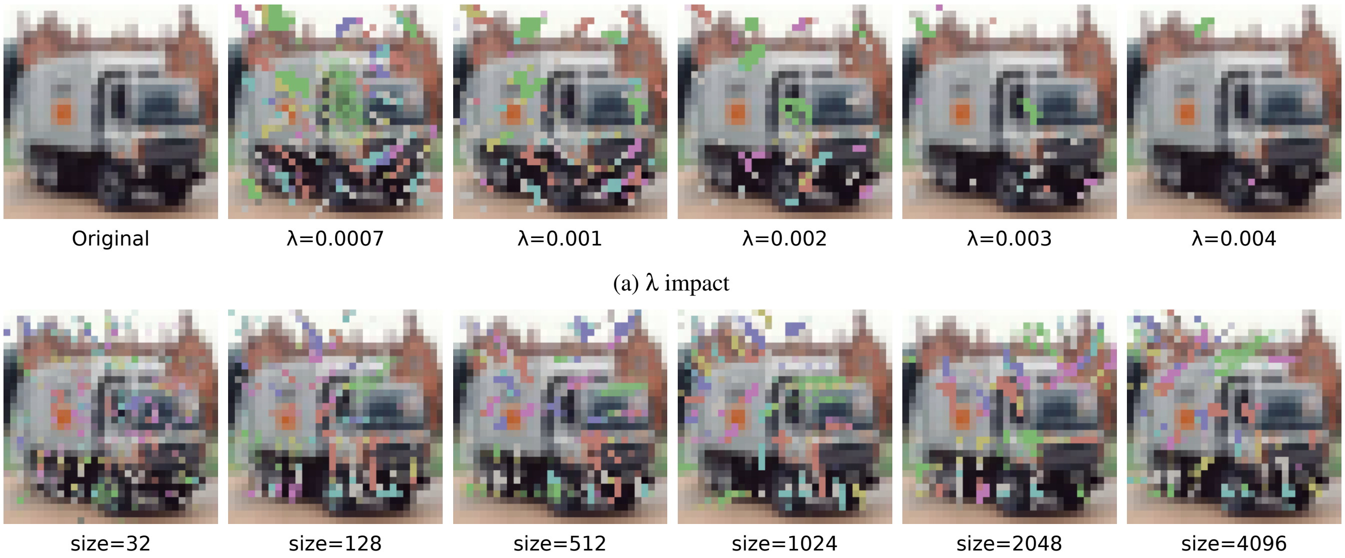
| Dataset/Model | ACC (%) | BAD (%) | ASR (%) |
|---|---|---|---|
| CIFAR-10/ResNet-18-V | 94.46 | 0.01±0.02 | 99.93±0.04 |
| GTSRB/VGG-16-V | 96.09 | 0.02±0.04 | 99.98±0.04 |
| CIFAR-10/VGG-16 | 89.31 | 0.01±0.03 | 99.91±0.02 |
| ImageNet/ResNet-18 | 68.79 | 0.06±0.01 | 99.84±0.13 |

### 6.7.3 Ablation Study: Trigger Generation and Bit Flips

In this section, we evaluate the impact of two critical modules in OneFlip: Trigger Generation and Bit Flipping on the attack performance of OneFlip. Specifically, we test two scenarios: 1) using only the Trigger Generation module without performing bit flipping (denoted as "Trigger") and 2) using only the Bit-Flipping module without applying a trigger to the input (denoted as "Bit-Flipping"). For each scenario, we calculate the average ASR under the potential weight set. The results, presented in Table 7, indicate that when either the Trigger Generation module or the Bit-Flipping module is used in isolation, the maximum ASR achieved by OneFlip does not exceed 15%. Moreover, as the number of classes in the dataset increases, the ASR drops below 1%, highlighting the limited effectiveness of either module on its own. In contrast, combining the Trigger Generation module with the Bit-Flipping module yields an ASR approaching 100%. These results demonstrate that both modules are essential and complementary for achieving the high ASR observed in OneFlip.

### 6.7.4 Ablation Study: More Dataset/Model Combinations

In this section, we evaluate OneFlip's performance on more dataset/model combinations to demonstrate that OneFlip is agnostic to classification accuracy, dataset choice, and model



(a) λ impact

(b) Sample size impact

Figure 5: The impact of different parameters on the visibility of generated triggers of OneFlip. Example images are from the CIFAR-10 test dataset.

architecture, thereby generalizing well across different settings. To examine the impact of classification accuracy, we use ResNet-18 and VGG-16 variants tailored for 32×32 images to obtain a better accuracy on CIFAR-10 and GTSRB, respectively. To assess the influence of dataset and architecture, we include CIFAR-10/VGG-16 and ImageNet/ResNet-18 in our evaluation. For ImageNet/ResNet-18, we use the pre-trained model released by PyTorch. Other combinations are trained from scratch using the same training settings as described in Section 6.1. The attack configuration also follows Section 6.1. As shown in Table 8, OneFlip consistently achieves high attack performance, indicating that it is not specific to model accuracy, dataset, or architecture. This ablation study confirms that OneFlip remains effective across a wide range of DNN models.

## 7 Discussions on Defenses

In this section, we analyze the resistance of OneFlip against backdoor defenses. Common backdoor defense methods include backdoor detection, backdoor mitigation, and input filtering. These methods allow the victim user to identify or neutralize compromised models or filter malicious inputs during training or deployment.

**Resistance to Detection.** Backdoor detection methods aim to identify target classes of backdoor attacks by reverse engineering triggers and applying outlier detection on those triggers [13,30,32,53,73,78,80]. Neural Cleanse [78], for example, employs gradient descent to reverse engineer triggers for each class of the model and uses the Median Absolute Deviation (MAD) algorithm to detect outliers in the $L_1$ norm of these triggers. Labels associated with outliers are flagged as potential backdoor target labels. (After detecting a backdoored model, users will attempt to mitigate the backdoor by fine-tuning the model [47,50,55,59,63,82].)

However, one major limitation of these defenses is that they can only detect the backdoor when the backdoor is injected during the training process or in the supply chain [2,6,14,68,88]. They cannot effectively defend against inference-time backdoor injection during runtime when the inference has already started. As a result, OneFlip, as an inference-time backdoor injection method, can be considered practically immune to such defenses. Additionally, these detection methods can also be conducted during the inference stage, similar to other integrity checking techniques. However, they are typically applied periodically, allowing an attacker to flip a bit between checks. Moreover, performing frequent integrity checks introduces computational overhead in terms of both CPU usage and memory bandwidth consumption.

**Resistance to Mitigation.** Victims may still want to retrain the model, which can be effective against inference-time backdoor injection methods, as it modifies the model's parameters, preventing the attacker from knowing the updated weights, thereby rendering bit flipping ineffective. To evaluate OneFlip's resistance to retraining, we further train the target models for 5 epochs on all datasets using the corresponding training dataset with a learning rate of 0.0001.

In response, the attacker employs an adaptive strategy. Beyond flipping the originally designated bit, the attacker systematically flips each of the other 6 lower bits in the same

Table 9: The benign accuracy (BA), and the average and maximum attack success rate (ASR) for the retrained model using adaptive OneFlip on the entire weight and trigger pairs. The retrained models are collected at the end of each retraining epoch (e.g., E1 represents the retrained model after Epoch 1).

| Dataset | Metric | E1 | E2 | E3 | E4 | E5 |
|---|---|---|---|---|---|---|
| CIFAR-10 | BA (%) | 86.47 | 86.61 | 86.56 | 86.74 | 86.49 |
|  | Mean ASR (%) | 13.31 | 13.31 | 13.29 | 13.30 | 13.29 |
|  | Max ASR (%) | 99.88 | 99.92 | 99.89 | 99.87 | 99.88 |
| GTSRB | BA (%) | 90.79 | 90.78 | 90.89 | 90.86 | 90.84 |
|  | Mean ASR (%) | 5.28 | 5.26 | 5.25 | 5.24 | 5.23 |
|  | Max ASR (%) | 99.47 | 99.34 | 99.25 | 99.16 | 99.06 |
| CIFAR-100 | BA (%) | 75.29 | 75.35 | 75.22 | 75.23 | 75.27 |
|  | Mean ASR (%) | 1.41 | 1.41 | 1.40 | 1.40 | 1.40 |
|  | Max ASR (%) | 99.88 | 99.92 | 99.92 | 99.95 | 99.93 |
| ImageNet | BA (%) | 79.97 | 79.86 | 79.75 | 79.69 | 79.63 |
|  | Mean ASR (%) | 2.16 | 2.15 | 2.15 | 2.15 | 2.15 |
|  | Max ASR (%) | 99.11 | 99.01 | 98.97 | 98.93 | 98.89 |

Table 10: The difference in the average neuron output of the retrained model compared to the original target model when inputting test samples from the test set added with OneFlip-generated triggers. The retrained models are collected at the end of each retraining epoch (e.g., E1 represents the retrained model after Epoch 1).

| Dataset | Neuron Output Difference |  |  |  |  |
|---|---|---|---|---|---|
|  | E1 | E2 | E3 | E4 | E5 |
| CIFAR-10 | 0.0009 | 0.0006 | 0.002 | 0.002 | 0.003 |
| GTSRB | 0.004 | 0.011 | 0.017 | 0.020 | 0.023 |
| CIFAR-100 | 0.071 | 0.057 | 0.025 | 0.005 | 0.028 |
| ImageNet | 0.611 | 0.707 | 0.770 | 0.842 | 0.891 |

weight's exponent (excluding the highest bit) one by one. This adaptive approach is feasible because 32-bit floating-point weights are stored contiguously in memory. Knowing the target bit's location allows the attacker to easily identify and manipulate adjacent bits of the same weight. For each fine-tuning epoch, we measured the BA of the fine-tuned model and the average and maximum ASR on the fine-tuned model using the adaptive OneFlip across all weight-trigger pairs. The results in Table 9 demonstrate that while backdoor mitigation significantly reduces OneFlip's average ASR to nearly unusable levels, the attacker can nevertheless identify bit-trigger pairs that achieve an ASR close to 100%, posing a continued threat to DNNs.

To investigate why OneFlip can still work even when the model's parameters are modified, we input the test samples from the test set added with OneFlip-generated triggers into these retrained models and compute the average neuron output. We then calculate the difference in the average neuron output compared to the target model. The experimental results, shown in Table 10, highlight the transferability of OneFlip-generated triggers. Although retraining alters the model's weight parameters, the triggers generated by OneFlip remain effective, continuing to significantly increase the corresponding neuron output in retrained models with minimal difference. Furthermore, adaptive OneFlip sequentially flips the other 6 non-MSB bits (excluding the originally identified bit), increasing the chances of encountering a new eligible pattern in the retrained model's weights. This ensures that the Bit-Flipping module remains operational. Consequently, adaptive OneFlip can still identify bit-trigger pairs that achieve very high attack success rates on fine-tuned models, as evidenced by the Max ASR in Table 9.

**Resistance to Filtering.** Input filtering methods aim to detect and filter malicious samples during the training or inference phases of a model [15, 31, 38, 56, 90, 91]. Filtering methods employed during the inference phase may pose a challenge to OneFlip, as it relies on feeding trigger-embedded inputs to the model during deployment to execute the attack. Nevertheless, many existing works have proposed highly stealthy trigger designs for backdoor attacks [40, 81]. These stealthy methods can be integrated into OneFlip to enhance its ability to evade detection and filtering during the inference phase, further improving its robustness against filtering-based defenses.

## 8 Conclusion

We present OneFlip, the first inference-time backdoor attack that requires only a single bit flip, significantly enhancing the practicality of backdoor attacks. Unlike previous inference-time backdoor attacks, OneFlip is also the first approach effective on full-precision models. It relies on a novel workflow that first identifies potential weights and then generates effective triggers, ensuring that the altered weight is activated by the trigger. The attack is demonstrated on DDR3 and DDR4. We evaluate OneFlip on various DNN architectures, including a vision transformer. The results show that OneFlip significantly outperforms prior methods, achieving near-perfect attack success rates (up to 99.9%, with an average of 99.6%) while causing minimal degradation to benign accuracy (as low as 0.005%, averaging 0.06%). Our work reveals a critical hardware-based vulnerability in DNNs: a highly effective backdoor can be injected into a full-precision model through a single bit flip. This highlights the need for robust defenses to safeguard deep learning applications from such attacks.

## Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-2304720 and CNS-2309477. It was also supported in part by the Commonwealth Cyber Initiative (CCI). The authors extend their gratitude to the anonymous reviewers and shepherd for their invaluable feedback and suggestions.

## Ethics Considerations

Our paper proposes a novel inference-stage backdoor attack, OneFlip, which injects a backdoor into a full-precision model via a single bit flip. This work aims to advance the understanding of model vulnerabilities and support the development of more robust machine learning systems. However, we acknowledge the potential security risks of such research. OneFlip may threaten stakeholders such as MLaaS providers on multi-tenant platforms and their users. A successful attack could compromise models of MLaaS, damage the reputation of their providers and users, and pose public safety risks in applications like autonomous driving or medical imaging. We carefully evaluated the likelihood of real-world harm from OneFlip and identified several factors that mitigate its practical threat level. First, OneFlip assumes white-box access, meaning the attacker must obtain the target model, while many companies keep their models confidential. Second, the attacker-controlled process must reside on the same physical machine as the target model, which may be difficult to achieve. Overall, we conclude that while the theoretical risks are non-negligible, the practical risk remains low, and the potential benefit of exposing previously unknown vulnerabilities outweighs these concerns. All experiments of the study were conducted on publicly available datasets and models within a private execution environment. Additionally, we note that Rowhammer is a publicly known hardware vulnerability; our work does not introduce new attack vectors at the hardware level, and thus, no additional disclosure is warranted.

## Open Science

The source code, datasets, representative trained models, configuration files, and experimental scripts are publicly released via Zenodo at `https://zenodo.org/records/15612334`. The release includes detailed instructions for installation and execution, as well as examples of victim models and models backdoored by OneFlip, as discussed in the paper. These resources are intended to support and enhance the reproducibility and replicability of our scientific findings.

## References

[1] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *IEEE International On-Line Testing Symposium*, 2010.

[2] Sabbir Ahmed, Ranyang Zhou, Shaahin Angizi, and Adnan Siraj Rakin. Deep-troj: An inference stage trojan insertion algorithm through efficient weight replacement attack. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems*, 2019.

[4] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via overparameterization. In *International Conference on Machine Learning*, 2019.

[5] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, 2019.

[6] Jiawang Bai, Kuofeng Gao, Dihong Gong, Shu-Tao Xia, Zhifeng Li, and Wei Liu. Hardly perceptible trojan attack against neural networks with bit flips. In *European Conference on Computer Vision*, 2022.

[7] Jiawang Bai, Baoyuan Wu, Yong Zhang, Yiming Li, Zhifeng Li, and Shu-Tao Xia. Targeted attack against deep neural networks via flipping limited weight bits. In *International Conference on Learning Representations*, 2021.

[8] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

[9] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in CNNS by training set corruption without label poisoning. In *IEEE International Conference on Image Processing*, 2019.

[10] Andrew Boutros, Mathew Hall, Nicolas Papernot, and Vaughn Betz. Neighbors from hell: Voltage attacks against deep learning accelerators on multi-tenant fpgas. In *International Conference on Field-Programmable Technology*, 2020.

[11] Kunbei Cai, Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. Deepvenom: Persistent DNN backdoors exploiting transient weight perturbations in memories. In *IEEE Symposium on Security and Privacy*, 2024.

[12] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[13] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *International Joint Conference on Artificial Intelligence*, 2019.

[14] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *International Conference on Computer Vision*, 2021.

[15] Weixin Chen, Baoyuan Wu, and Haoqian Wang. Effective backdoor defense by exploiting sensitivity of poisoned samples. In *Advances in Neural Information Processing Systems*, 2022.

[16] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR*, abs/1712.05526, 2017.

[17] Yanzuo Chen, Zhibo Liu, Yuanyuan Yuan, Sihang Hu, Tianxiang Li, and Shuai Wang. Compiled models, built-in exploits: Uncovering pervasive bit-flip attack surfaces in dnn executables. In *Network and Distributed System Security Symposium*, 2025.

[18] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In *IEEE International Symposium on Hardware Oriented Security and Trust*, 2019.

[19] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: synchronized many-sided rowhammer attacks from javascript. In *USENIX Security Symposium*, 2021.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2009.

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

[22] Jianshuo Dong, Han Qiu, Yiming Li, Tianwei Zhang, Yuanjie Li, Zeqi Lai, Chao Zhang, and Shu-Tao Xia. One-bit flip is all you need: When bit-flip attack meets model training. In *IEEE/CVF International Conference on Computer Vision*, 2023.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[24] Ali Fakhrzadehgan, Yale N. Patt, Prashant J. Nair, and Moinuddin K. Qureshi. Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection. In *IEEE International Symposium on High-Performance Computer Architecture*, 2022.

[25] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Trrespass: Exploiting the many sides of target row refresh. In *IEEE Symposium on Security and Privacy*, 2020.

[26] Dennis R. E. Gnad, Fabian Oboril, and Mehdi Baradaran Tahoori. Voltage drop-based fault attacks on fpgas using valid bitstreams. In *International Conference on Field Programmable Logic and Applications*, 2017.

[27] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[28] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *IEEE Symposium on Security and Privacy*, 2018.

[29] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 2019.

[30] Junfeng Guo, Ang Li, and Cong Liu. AEVA: black-box backdoor detection using adversarial extreme value analysis. In *International Conference on Learning Representations*, 2022.

[31] Junfeng Guo, Yiming Li, Xun Chen, Hanqing Guo, Lichao Sun, and Cong Liu. SCALE-UP: an efficient black-box input-level backdoor detection via analyzing scaled prediction consistency. In *International Conference on Learning Representations*, 2023.

[32] Wenbo Guo, Lun Wang, Yan Xu, Xinyu Xing, Min Du, and Dawn Song. Towards inspecting and eliminating trojan backdoors in deep neural networks. In *IEEE International Conference on Data Mining*, 2020.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.

[35] Hokuto Hirano and Kazuhiro Takemoto. Simple iterative method for generating targeted universal adversarial perturbations. *Algorithms*, 2020.

[36] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *USENIX Security Symposium*, 2019.

[37] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *International Joint Conference on Neural Networks*, 2013.

[38] Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, and James Bailey. Distilling cognitive backdoor patterns within an image. In *International Conference on Learning Representations*, 2023.

[39] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: scalable rowhammering in the frequency domain. In *IEEE Symposium on Security and Privacy*, 2022.

[40] Wenbo Jiang, Hongwei Li, Guowen Xu, and Tianwei Zhang. Color backdoor: A robust poisoning attack in color space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[41] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and rowpress without physical address information. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2024.

[42] William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 1996.

[43] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE International Symposium on Computer Architecture*, 2014.

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

[45] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. Rambleed: Reading bits in memory without accessing them. In *IEEE Symposium on Security and Privacy*, 2020.

[46] Shaofeng Li, Xinyu Wang, Minhui Xue, Haojin Zhu, Zhi Zhang, Yansong Gao, Wen Wu, and Xuemin (Sherman) Shen. Yes, one-bit-flip matters! universal DNN model inference depletion with runtime code fault injection. In *USENIX Security Symposium*, 2024.

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

[48] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, 2018.

[49] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *IEEE/CVF International Conference on Computer Vision*, 2021.

[50] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses*, 2018.

[51] Wenye Liu, Chip-Hong Chang, Fan Zhang, and Xiaoxuan Lou. Imperceptible misclassification attack on deep learning accelerator by glitch injection. In *ACM/IEEE Design Automation Conference*, 2020.

[52] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *IEEE/ACM International Conference on Computer-Aided Design*, 2017.

[53] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. ABS: scanning neural networks for back-doors by artificial brain stimulation. In *ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[54] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Network and Distributed System Security Symposium*, 2018.

[55] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *IEEE International Conference on Computer Design*, 2017.

[56] Wanlun Ma, Derui Wang, Ruoxi Sun, Minhui Xue, Sheng Wen, and Yang Xiang. The "beatrix" resurrections: Robust backdoor detection via gram matrices. In *Network and Distributed System Security Symposium*, 2023.

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

[58] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[59] Bingxu Mu, Zhenxing Niu, Le Wang, Xue Wang, Qiguang Miao, Rong Jin, and Gang Hua. Progressive backdoor erasing via connecting backdoor and adversarial attacks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[60] Janani Mukundan, Hillery C. Hunter, Kyu-Hyoun Kim, Jeffrey Stuecheli, and José F. Martínez. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. In *Annual International Symposium on Computer Architecture*, 2013.

[61] Tuan Anh Nguyen and Anh Tuan Tran. Wanet - imperceptible warping-based backdoor attack. In *International Conference on Learning Representations*, 2021.

[62] Marco Oliverio, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Secure page fusion with vusion: https://www.vusec.net/projects/vusion. In *Symposium on Operating Systems Principles*, 2017.

[63] Lu Pang, Tao Sun, Haibin Ling, and Chao Chen. Backdoor cleansing with unlabeled data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[64] Xiangyu Qi, Tinghao Xie, Yiming Li, Saeed Mahloujifar, and Prateek Mittal. Revisiting the assumption of latent separability for backdoor defenses. In *International Conference on Learning Representations*, 2023.

[65] Han Qiu, Yi Zeng, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. Deepsweep: An evaluation framework for mitigating DNN backdoor attacks using data augmentation. In *ACM Asia Conference on Computer and Communications Security*, 2021.

[66] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *IEEE Symposium on Security and Privacy*, 2022.

[67] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[68] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. TBT: targeted neural network attack with bit trojan. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[69] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-BFA: targeted bit-flip adversarial weight attack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[70] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant FPGA. In *USENIX Security Symposium*, 2021.

[71] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *USENIX Security Symposium*, 2016.

[72] Gururaj Saileshwar, Bolin Wang, Moinuddin K. Qureshi, and Prashant 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.

[73] Guangyu Shen, Yingqi Liu, Guanhong Tao, Shengwei An, Qiuling Xu, Siyuan Cheng, Shiqing Ma, and Xiangyu Zhang. Backdoor scanning for deep neural networks through k-arm optimization. In *International Conference on Machine Learning*, 2021.

[74] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[75] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *USENIX Annual Technical Conference*, 2018.

[76] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. Spechammer: Combining spectre and rowhammer for new speculative attacks. In *IEEE Symposium on Security and Privacy*, 2022.

[77] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[78] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE Symposium on Security and Privacy*, 2019.

[79] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[80] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *European Conference on Computer Vision*, 2020.

[81] Tong Wang, Yuan Yao, Feng Xu, Shengwei An, Hanghang Tong, and Ting Wang. An invisible black-box backdoor attack through frequency domain. In *European Conference on Computer Vision*, 2022.

[82] Dongxian Wu and Yisen Wang. Adversarial neuron pruning purifies backdoored deep models. In *Advances in Neural Information Processing Systems*, 2021.

[83] Chunwei Xia, Jiacheng Zhao, Huimin Cui, Xiaobing Feng, and Jingling Xue. Dnntune: Automatic benchmarking dnn models for mobile-cloud computing. *ACM Transactions on Architecture and Code Optimization*, 2019.

[84] Pengfei Xia, Ziqiang Li, Wei Zhang, and Bin Li. Data-efficient backdoor attacks. In *International Joint Conference on Artificial Intelligence*, 2022.

[85] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *USENIX Security Symposium*, 2016.

[86] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *USENIX Security Symposium*, 2020.

[87] Zhi Zhang, Yueqiang Cheng, Minghua Wang, Wei He, Wenhao Wang, Surya Nepal, Yansong Gao, Kang Li, Zhe Wang, and Chenggang Wu. Softtrr: Protect page tables against rowhammer attacks using software-only target row refresh. In *USENIX Annual Technical Conference*, 2022.

[88] Mengxin Zheng, Qian Lou, and Lei Jiang. Trojvit: Trojan insertion in vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[89] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Gradient descent optimizes over-parameterized deep relu networks. *Machine learning*, 2020.

[90] Fei Zuo, Bokai Yang, Xiaopeng Li, and Qiang Zeng. Exploiting the inherent limitation of L0 adversarial examples. In *International Symposium on Research in Attacks, Intrusions and Defenses*, 2019.

[91] Fei Zuo and Qiang Zeng. Exploiting the sensitivity of L2 adversarial examples to erase-and-restore. In *ACM Asia Conference on Computer and Communications Security*, 2021.