Your Scale Factors are My Weapon: Targeted Bit-Flip Attacks on Vision Transformers via Scale Factor Manipulation

Jialai Wang[1,2*], Yuxiao Wu[3], Weiye Xu[4], Yating Huang[3], Chao Zhang[2],
Zongpeng Li[2†], Mingwei Xu[2], and Zhenkai Liang[1]
[1]National University of Singapore, [2]Tsinghua University,
[3]Huazhong University of Science and Technology, [4]China Mobile Research Institute

Abstract

Vision Transformers (ViTs) have experienced significant progress and are quantized for deployment in resource-constrained applications. Quantized models are vulnerable to targeted bit-flip attacks (BFAs). A targeted BFA prepares a trigger and a corresponding Trojan/backdoor, inserting the latter (with RowHammer bit flipping) into a victim model, to mislead its classification on samples containing the trigger. Existing targeted BFAs on quantized ViTs are limited in that: (1) they require numerous bit-flips, and (2) the separation between flipped bits is below 4 KB, making attacks infeasible with RowHammer in real-world scenarios. We propose a new and practical targeted attack `Flip-S` *against quantized ViTs. The core insight is that in quantized models, a scale factor change ripples through a batch of model weights. Consequently, flipping bits in scale factors, rather than solely in model weights, enables more cost-effective attacks. We design a Scale-Factor-Search (*`SFS`*) algorithm to identify critical bits in scale factors for flipping, and adopt a mutual exclusion strategy to guarantee a 4 KB separation between flips. We evaluate* `Flip-S` *on CIFAR-10 and ImageNet datasets across five ViT architectures and two quantization levels. Results show that* `Flip-S` *achieves attack success rate (ASR) exceeding* 90.0% *on all models with 50 bits flipped, outperforming baselines with ASR typically below* 80.0%*. Furthermore, compared to the SOTA,* `Flip-S` *reduces the number of required bit-flips by* $8\times$-$20\times$ *while reaching equal or higher ASR. Our source code is publicly available*[1].

1. Introduction

Recent developments in Vision Transformers (ViTs) have shown promise in and beyond vision-related tasks [12, 22]. To enable ViTs in resource-constrained applications, model quantization is customarily adopted [9, 10, 14, 43], converting model weights from high- to low-precision (*e.g.* float32 to int8), reducing memory usage and computational cost [3, 10, 17, 40]. Recent literature reveals that bit-flip attacks (BFAs) [44, 45] pose threats to quantized ViTs. BFAs tamper with model parameters [7, 27–29, 31, 39] for malicious attempts. They typically use RowHammer [19, 23] to exploit the hardware layer to introduce errors in memory regions that store a model's weight parameters, compromising the model's integrity and performance [24]. For instance, flipping a few critical bits of off-the-shelf DNN model parameters can mislead the inference to a target class.

Current BFAs can be categorized into untargeted and targeted attacks. *Untargeted* BFAs aim to compromise model accuracy to random guess level [27, 29, 39], *e.g.*, degrading accuracy from 80.4% to 0.2%. In contrast, *targeted* BFAs are more stealthy [4, 5, 7, 28, 44]. They mislead models to a target class on specific samples, *e.g.*, samples with a specific trigger. This work focuses on the widely discussed targeted BFAs. Such attacks typically contain two steps: first, generating a trigger, and then inserting a Trojan into the victim model by flipping specific bits. Once the Trojan is inserted, the victim model will misclassify any input containing the trigger as the target class.

We identify two main challenges in attacking quantized ViTs. First, existing attacks require flipping numerous bits, and are hence cost prohibitive. Our experiments reveal that targeted BFAs typically require flipping hundreds of bits to achieve a successful targeted attack on ViTs. Flipping such large number of bits is time-consuming. BFAs rely on RowHammer to conduct bit-flips in memory pages, which is not a trivial task. Previous literature [18, 36] shows that flipping as few as 24 bits can take several hours. Second, existing BFAs commonly flip bits whose separation is below 4 KB, limiting the practicality of these attacks. Prior work [33] suggests that RowHammer generally flips only a single bit within a 4 KB memory page. Since most BFAs

*The main work was conducted when the author studied at Tsinghua.
†Corresponding author.
[1]https://github.com/wjl123wjl/Flip-S

require multiple bit-flips within the same page, RowHammer's constraints reduce the feasibility of these attacks.

Observation. We observe that during quantized model inference, to maintain accuracy, low-precision weights are restored to high-precision using *scale factors*. A single scale factor change impacts a batch of model weights. This indicates that flipping a small number of bits in scale factors can effectively attack models. To this end, we primarily target scale factors for bit-flips, unlike previous works that focus on model weights.

We propose `Flip-S`, a novel and practical targeted bit-flip attack targeting quantized ViTs. `Flip-S` can effectively attack ViTs with significantly fewer bit-flips than existing literature, and guarantee 4KB-separation between flipped bits. We first design a trigger generation method. We then design a Scale-Factor-Search (`SFS`) algorithm to identify and flip critical bits in scale factors to insert a Trojan. Furthermore, we propose a mutual exclusion strategy that guarantees no more than one bit is flipped within any 4 KB separation. We compare `Flip-S` with four SOTA targeted BFAs on CIFAR-10 and ImageNet datasets across five ViT architectures and two quantization levels. Results show that `Flip-S` significantly outperforms the competing alternatives. For instance, to achieve a similar attack success rate (ASR), `Flip-S` flips $8\times$- $20\times$ fewer bits.

In summary, this work makes the following contributions.

- We propose a new and practical targeted BFA solution against quantized ViTs. Our `Flip-S` achieves effective attacks and outperforms all the baselines.
- We are the first to identify that scale factors can be exploited for bit-flip attacks, introducing a new dimension in attack design.
- `Flip-S` is more practical than existing targeted BFAs by requiring significantly fewer bit-flips and guarantees a separation of at least 4 KB between flipped bits, while the previous art cannot.

2. Background and Related Work

Bit-flip attacks. BFAs compromise models by flipping bits in model weights. They employ specific algorithms to identify critical bits and flip them (*i.e.* $0 \rightarrow 1$ and $1 \rightarrow 0$). Existing BFAs can be categorized into untargeted [27] and targeted attacks [4, 5, 7, 28]. Untargeted attacks aim to significantly degrade model accuracy, often reducing it to the level of random guessing. In contrast, targeted attacks are generally more stealthy, as they have a smaller impact on model accuracy. They mislead models to misclassify specific inputs or inputs containing a particular trigger into a designated target class.

Most targeted BFAs focus on attacking CNNs rather than ViTs. The workflow typically involves first generating a trigger and then inserting a Trojan into victim models through bit flips. TBT [28] first selects critical network neurons with significant influence on the target class, then tailor a specific trigger to activate these neurons. This solution formalizes an optimization problem to modify critical bits corresponding to these neurons. ProFlip [7] selects neurons through forwarding derivative-based saliency map construction [26], and generates triggers to stimulate salient neurons to large values. It then designs an efficient retrieval algorithm to flip critical bits. TA-LBF [4] does not require a trigger and instead misclassifies specific samples to a target class. We do not include it in our comparisons. HPT [5] builds on TA-LBF by reconsidering the use of triggers in attacks. Tol *et al.* [33] propose a targeted BFA that guarantees no more than one bit-flip in a memory page, but requires nearly 1400 bit-flips on ImageNet dataset.

A few studies [3, 44, 45] attempt to attack ViTs. Zhou *et al.* [45] adapt an untargeted attack [27], originally designed for CNNs, to ViTs. Deep-TROJ [3] flips bits in page frame number to replace a target weight block with another. However, we find it struggles to achieve a high ASR even when flipping numerous bits. Zheng *et al.* [44] propose a targeted attack solution TrojViT. They focus on designing patch-wise triggers rather than identifying critical bits for flipping. These studies require flipping numerous bits to attack ViTs. For instance, TrojViT flips nearly 300 bits to achieve a 92.0% ASR on ViT-base [38] model.

Model quantization and scale factors. Existing BFAs primarily target quantized models, and we discover that flipping bits in scale factors can effectively attack quantized models. We introduce backgrounds on model quantization and scale factors. Model quantization techniques are widely used in deep learning systems, especially for resource-constrained applications [15, 39]. Quantization transforms model weights from high-precision floating-point numbers (*e.g.* float32) to low-precision numbers (*e.g.* int8). This transformation reduces both memory usage and computational cost [3, 10, 17, 40].

However, the reduced precision in model weights comes with accuracy degradation. To mitigate this, many quantization techniques apply dequantization, converting low-precision weights back to high-precision values as they are involved in computations. Dequantization relies on *scale factors*. For example, given a scale factor S, dequantization from int8 to float32 can be expressed as:

$$w_{\text{dequantized}} = Q(w) \times S, \tag{1}$$

where w is the original weight in float32, $Q(w)$ is its corresponding quantized weight in int8, and $w_{\text{dequantized}}$ is the dequantized weight in float32.

Many quantization solutions [9, 10, 14, 43] segment model weights into different groups, assigning a scale factor to each group. For instance, as shown in Figure 1, the quantized weights $\{Q(w_1), \ldots, Q(w_i)\}$ use scale factor S_1 for dequantization, while the quantized weights $\{Q(w_{i+1}), \ldots,$

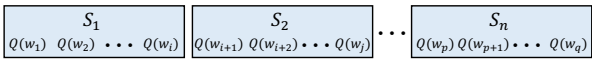

Figure 1. Model weights are segmented into different groups, each assigned a scale factor. For example, the quantized weights $\{Q(w_1), \ldots, Q(w_i)\}$ use scale factor S_1 for dequantization, while the quantized weights $\{Q(w_{i+1}), \ldots, Q(w_j)\}$ use S_2.

$Q(w_j)\}$ use scale factor S_2. *This indicates that flipping bits in one scale factor can impact a batch of model weights.* To this end, our work incorporates bit-flips in scale factors, whereas existing BFAs are limited to bit-flips in model weights.

3. Problem Definition

Threat model. Our threat model follows prior art [3, 7, 28, 33, 39, 44]. The adversary targets quantized models and is aware of the weight quantization methods. The adversary knows the architecture and weights of the models, and can implement precise bit-flips at desired locations. Furthermore, the adversary has a small set of test data, which can be easily collected and labeled by the adversary since the task of the target model is known. Note that the adversary does not require information of the training data or access to the training pipeline.

Adversarial goals and requirements. We focus on targeted attacks, with adversary goals consistent with most previous studies [5, 7, 27, 28, 36, 44]. The adversary designs a specific trigger and injects the corresponding Trojan into models via bit flips. For any input sample containing this trigger, the Trojaned models will misclassify it as the target class.

The adversary should consider attack feasibility, which hinges on two primary factors: the total number of bit-flips and the separation between flipped bits. First, the number of bit-flips should be minimized. Flipping bits necessitates RowHammer. RowHammer techniques achieve bit-flips by repeatedly accessing (or "hammering") a specific DRAM row, which causes interference effects that flip bits in adjacent rows. This process is time-consuming. For instance, flipping 50 bits takes several hours [18, 36]. Second, the separation between flipped bits should be at least 4 KB. Previous studies [33] show that RowHammer typically induces no more than one bit-flip within a 4 KB memory page. *Unfortunately, our experiments reveal that current BFAs do not reliably meet these requirements when applied to ViTs.*

4. Methodology

We propose a new, practical targeted bit-flip attack, `Flip-S`. The first step is trigger generation. The second step is to identify critical bits and flip them to insert a Trojan into the victim model. An input containing the trigger activates the Trojan, causing the model to misclassify the input into a specified target class. We show an intuitive example in Figure 2.

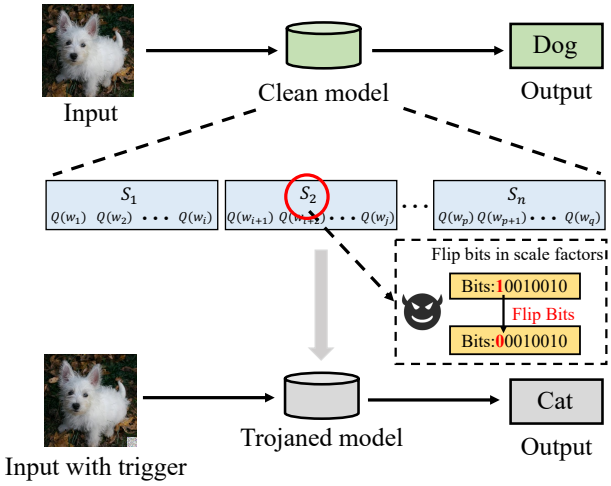

Figure 2. Effect of `Flip-S`. The top section illustrates standard inference on a clean model, while the bottom section shows that after critical bits flipped, the Trojaned model produces an incorrect output when the input contains a trigger.

4.1. Trigger Generation

Given that ViTs are based on attention mechanisms, the basic idea behind our trigger generation is to maximize the attention directed toward the generated trigger across layers. Formally, given a batch of test input samples x, we define their trigger-embedded versions as $\hat{x} = (1-m) \cdot x + m \cdot t$, where t represents the trigger and m is a mask that specifies the trigger's location within each sample. Notably, we do not require access to training data, while a small set of test samples is sufficient. Since each layer in a ViT includes an attention mechanism, we aim to maximize the attention the trigger receives across all layers. For each layer i in a ViT, we define the attention of the trigger in current layer as:

$$A_i(\hat{x}) = \sum_h attn(i)_j^h, \tag{2}$$

where $A_i(\hat{x})$ represents the total attention weight allocated to trigger position j in layer i, summed over all attention heads. Here, $attn(i)$ is the raw attention weight matrix in the i-th layer. $attn(i)_j^h$ denotes the attention weight assigned to the trigger by the h-th head in the i-th layer, quantifying the contribution of each head to the overall attention. The summation $\sum_h$ aggregates the attention weights across all heads.

As the model goes deeper, the attention weights between different tokens tend to become more similar [2]. This occurs because the contextualized embeddings increasingly

resemble each other in deeper layers, leading to a uniform distribution of attention across tokens. Such uniformity hinders our goal of maximizing attention directed specifically at the trigger position. To mitigate this issue, we refer to attention rollout [2], which recursively accumulates attention weights up to the current layer. Formally, attention rollout is defined as:

$$\widetilde{attn}(i) = \begin{cases} attn(i)\widetilde{attn}(i-1), & \text{if } i > 1 \\ attn(i), & \text{if } i = 1 \end{cases} \tag{3}$$

To this end, we replace the original raw attention matrix $attn(i)$ in Eq. (2) with the attention rollout $\widetilde{attn}(i)$, and the $A_i(\hat{x})$ can be redefined as:

$$A_i(\hat{x}) = \sum_h \widetilde{attn}(i)^h_j. \tag{4}$$

To maximize the cumulative attention directed at the trigger position across all layers, we strive to increase the aggregated attention values $A_i(\hat{x})$ as much as possible. To this end, we define an optimization objective that encourages each $A_i(\hat{x})$ to approximate a high attention level, denoted by a target constant α. By minimizing the squared deviation between $A_i(\hat{x})$ and α, we enforce the attention at the trigger position to be consistently high across all layers. We formalize the optimization problem as follows:

$$\mathcal{L}_{atten} = \sum_{i=1}^{N} \|A_i(\hat{x}) - \alpha\|_2^2, \tag{5}$$

where N is the total number of layers. The problem then becomes ensuring the trigger-embedded samples $\hat{x}$, classified into a target class y. We utilize the cross-entropy loss function $\mathcal{L}_{CE}(\hat{x}, y)$ to achieve this. The complete optimization objective for trigger generation is defined as:

$$\mathcal{L}(\hat{x}, y) = \mathcal{L}_{CE}(\hat{x}, y) + \lambda \mathcal{L}_{atten}(\hat{x}), \tag{6}$$

where λ is a hyperparameter. By minimizing $\mathcal{L}(\hat{x}, y)$, we optimize the trigger.

4.2. Critical Bit Identification

We design a gradient-based algorithm, Scale-Factor-Search (SFS), to identify critical bits in scale factors. Furthermore, we propose a mutual exclusion strategy to guarantee the separation between flipped bits.

4.2.1. Loss Function

We first define a loss function to guide SFS. The objective is to misclassify inputs containing the trigger as a target class, without affecting model accuracy on clean samples. Formally, given a batch of test input samples x, their trigger-embedded versions $\hat{x}$, a clean model f, and the compromised model $\hat{f}$, we define the targeted loss function as follows:

$$\mathcal{L}(\hat{x}, x, y) = \gamma \mathcal{L}_{CE}(\hat{f}(\hat{x}), y) + \|\hat{f}(x) - f(x)\|_2^2, \tag{7}$$

where y denotes the target class, and γ is a hyperparameter. The first term, $\mathcal{L}_{CE}(\hat{f}(\hat{x}), y)$, encourages the compromised model to misclassify $\hat{x}$ into the target class y, while the second term, $\|\hat{f}(x) - f(x)\|_2^2$, helps preserve model $\hat{f}$ accuracy on clean samples.

4.2.2. Scale-Factor-Search

We propose SFS to identify and flip critical bits within scale factors. The primary goal of SFS is to manipulate scale factors by calculating gradients of the loss function *w.r.t.* these factors to locate the most critical ones, then identifying which bits within them are most relevant to flip for reducing the loss. A key challenge arises because manipulating scale factors affects batches of weights, potentially causing significant changes that can trap the optimization process. To address this, SFS incorporates a fallback adjustment mechanism: if the optimization becomes trapped, SFS temporarily switches to flipping bits in weights. This provides finer-grained adjustments that help the process escape from the trap, after which SFS can return to flipping scale factor bits. The overall workflow of SFS is shown in Algorithm 1. SFS operates in two main phases:

- Scale factor bit-flip phase: SFS begins by computing gradients of the loss function *w.r.t.* scale factors to identify the critical ones. It then iteratively evaluates bits within these scale factors, selecting the bit whose flip would most effectively reduce the loss. If flipping the identified bit reduces the loss, SFS flips it and applies a mutual exclusion strategy to maintain adequate separation between flipped bits. The process then continues within this phase. If no bit-flip leads to a loss reduction, the optimization is trapped and SFS switches to the weight bit-flip phase.
- Weight bit-flip phase: SFS computes the gradients of the loss function *w.r.t.* weights to pinpoint critical weights. It then iteratively assesses bits within these weights, selecting the bit whose flip would most effectively reduce the loss, similar to the process in the scale factor bit-flip phase. After flipping this bit, SFS employs the mutual exclusion strategy to ensure separation between flipped bits. Consequently, the process returns to the scale factor bit-flip phase.

This iterative approach continues until the total number of bits flipped reaches a predefined threshold n_{max}.

Scale factor bit-flip phase. For simplicity, we denote the loss function defined in Eq. (7) as $\mathcal{L}$. Consider a set of scale factors $S = \{S_1, \ldots, S_n\}$ from the target model, where each scale factor $S_i \in S$ corresponds to a set of q weights, $w = \{w_1, \ldots, w_q\}$. Each weight $w_j \in w$ is defined as $w_j = Q(w_j) \times S_i$, where $Q(w_j)$ is the quantized

Algorithm 1: The workflow of SFS

Input: a batch of test input samples x, trigger-embedded versions $\hat{x}$, a target class y, a clean model f, and a predefined threshold n_{max};
Output: the compromised model $\hat{f}$;
$\hat{f} = f$; $n_b = 0$; Initialize M_S and M_w with 0;
while $n_b < n_{max}$ **do**
  $\mathcal{L} =$ CalculateLoss$(\hat{x}, x, y, f, \hat{f})$;
  $flag =$ Scale-Factor-Bit-Flip$(\mathcal{L}, M_S, \hat{f})$;
  if *flag* **then**
    Weight-Bit-Flip$(\mathcal{L}, M_w, \hat{f})$;
  $n_b = n_b + 1$;
return $\hat{f}$;
Function Scale-Factor-Bit-Flip$(\mathcal{L}, M_S, \hat{f})$**:**
  $S_{top} =$ Select-Topk-Scale-Factors$(\mathcal{L}, M_S, \hat{f})$;
  $R = 0$;// record the maximum reduction of the loss function
  $f_R = \hat{f}$;
  for S_t *in* S_{top} **do**
    for *bit in* S_t **do**
      $f_{temp} =$ Flip-Bit$(\hat{f}, bit)$;
      $\mathcal{L}_{temp} =$CalculateLoss$(\hat{x}, x, y, f, f_{temp})$;
      $\triangle\mathcal{L} = \mathcal{L} - \mathcal{L}_{temp}$;
      if $\triangle\mathcal{L} > R$ **then**
        $R = \triangle\mathcal{L}$;
        $f_R = f_{temp}$;
  if $R > 0$ **then**
    $\hat{f} = f_R$;
    Mutual-Exclusion-Strategy(M_s);
    return false;
  else
    return true;
Function Weight-Bit-Flip$(\mathcal{L}, M_w, \hat{f})$**:**
  $w_{top} =$ Select-Topη-Weights$(\mathcal{L}, M_w, \hat{f})$;
  $R = 0$;
  $f_R = \hat{f}$;
  for w_j *in* w_{top} **do**
    for *bit in* w_j **do**
      $f_{temp} =$ Flip-Bit$(\hat{f}, bit)$;
      $\mathcal{L}_{temp} =$CalculateLoss$(\hat{x}, x, y, f, f_{temp})$;
      $\triangle\mathcal{L} = \mathcal{L} - \mathcal{L}_{temp}$;
      if $\triangle\mathcal{L} > R$ **then**
        $R = \triangle\mathcal{L}$;
        $f_R = f_{temp}$;
  $\hat{f} = f_R$;
  Mutual-Exclusion-Strategy(M_w);

value of w_j. To compute the gradient of $\mathcal{L}$ with respect to S_i, we aggregate the gradients of $\mathcal{L}$ *w.r.t.* the associated weights. Formally, the gradient of $\mathcal{L}$ with respect to S_i is defined as:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial S_i} &= \sum_{j=1}^{q} \frac{\partial \mathcal{L}}{\partial w_j} \times \frac{\partial w_j}{\partial S_i} \times (M_S)_i \\ &= \sum_{j=1}^{q} \frac{\partial \mathcal{L}}{\partial w_j} \times Q(w_j) \times (M_S)_i,\end{aligned} \tag{8}$$

where $(M_S)_i \in \{0, 1\}$ is used to control separation between flipped bits, and more details are introduced in the mutual exclusion strategy. SFS relies on each gradient $\frac{\partial \mathcal{L}}{\partial S_i}$ to identify critical scale factors. Specifically, SFS selects top-k ($k = 50$ in our settings) scale factors with highest gradient magnitudes, denoted as S_{top}. For each scale factor $S_t \in S_{top}$, SFS iteratively evaluates each bit in S_t. After evaluating all bits within these scale factors, SFS flips the bit that minimizes $\mathcal{L}$ most effectively. If no bit-flip reduces $\mathcal{L}$, the optimization is trapped, and SFS switches to the weight bit-flip phase.

Weight bit-flip phase. In this phase, SFS flips bits in model weights rather than scale factors. This phase introduces finer-grained changes and thus helps the optimization escape from local minima. We denote the weight corpus as $w = \{w_1, \ldots, w_q\}$. SFS computes the gradient of $\mathcal{L}$ *w.r.t.* each $w_j \in w$, defined as $\frac{\partial \mathcal{L}}{\partial w_j} * (M_w)_j$, where $(M_w)_j$ controls separation between flipped bits. After that, SFS selects top-η ($\eta = 50$ in our settings) weights with the highest gradient magnitudes, denoted as w_{top}. For each weight $w_j \in w_{top}$, SFS iteratively examines each bit. After evaluating all bits within these weights, SFS flips the bit that achieves the greatest reduction in $\mathcal{L}$. Afterward, SFS returns to the scale factor bit-flip phase.

Mutual exclusion strategy. The question left is how to guarantee the separation between flipped bits. SFS employs a mutual exclusion strategy to achieve this goal. In the scale factor bit-flip phase, if a bit is flipped in a given scale factor, then that scale factor and all other scale factors within 4 KB are restricted from further flipping. This is represented by a binary control matrix $M_S = \{(M_S)_1, \ldots, (M_S)_n\}$, where each element $(M_S)_i$ corresponds to a scale factor S_i and is initialized to 1. If S_i cannot be flipped, $(M_S)_i$ is set to 0. Similarly, in the weight bit-flip phase, if a bit is flipped, that weight and all other weights within 4 KB are restricted from further flipping. SFS applies a control matrix $M_w = \{(M_w)_1, \ldots, (M_w)_q\}$, where each element $(M_w)_j$ corresponds to a weight w_j and is also initialized to 1. If w_j cannot be flipped, $(M_w)_j$ is set to 0.

5. Evaluation

We conduct experiments on a computer with an Intel Xeon Gold 6154 CPU and four NVIDIA Tesla V100 GPUs.

5.1. Experimental Setup

ViT models and datasets. We conduct experiments on 5 commonly used pre-trained ViT models, *i.e.*, ViT-base [38], Deit-base [34], Deit-small [34], SwinT-base [22] and SwinT-small [22], with two datasets, *i.e.*, CIFAR-10, and ImageNet. On ImageNet, we use their open-sourced pre-trained models. On CIFAR-10, the models are fine-tuned based on the pre-trained models on ImageNet.

Qunatization settings. Previous studies consider only 8-bit quantization level. Besides 8-bit quantization level, our work considers a more challenging scenario, *i.e.*, 4-bit quantization level. A lower quantization level provides fewer bits for flipping, which reduces attack space and increases the difficulty of attacks. To achieve these two quan-

tization levels, we adopt the widely used bitsandbytes library [1], which provides efficient quantization methods designed for models in resource-constrained environments.

Baselines and hyperparameters. We compare our `Flip-S` with four SOTA targeted BFAs, *i.e.*, TBT [28] ProFlip [7], HPT [5], and TrojViT [44]. We reproduce the baselines with their open-source code and the recommended parameters. Following TrojViT, for all solutions, we set the target class to 2, and the percentage of the trigger area in an input image, to 4.59%. All baselines require randomly sampling a few test data from the test/validation set to generate triggers and identify critical bits: On ImageNet, TorjViT requires 384 test data, while TBT, ProFlip and HPT require 256 test data. On CIFAR-10, all baselines require 128 test data. In contrast, our `Flip-S` only randomly samples 32 test data for CIFAR-10 and ImageNet respectively. We set α to 100 (in Eq. (5)), λ (in Eq. (6)) and γ (in Eq. (7)) to 1.

Bit-flip number limit. Following previous art [36], we set the bit-flip number limit (n_{max}) to 50 for all solutions. Previous studies [18, 36] indicate that achieving this level of bit-flips in real-world conditions requires at least several hours, rendering higher bit-flip counts impractical. With this constraint, we compare our `Flip-S` with baselines. To provide a comprehensive evaluation, we further relax n_{max} to 500 and re-evaluate all solutions.

Evaluation metrics. We use the following three metrics for evaluation. The first two are widely used in the literature [7, 28, 44], while the third is introduced to assess attack feasibility:

- *Attack success rate (ASR)*: The percentage of inputs embedded with a trigger that are classified into the predefined target class. A higher ASR indicates a more effective attack.
- *Clean data accuracy (CDA)*: The percentage of inputs without a trigger that are correctly classified into their respective classes. A higher CDA indicates a more stealthy attack.
- *4KB bit-flip separation compliance (BFSC)*: As described in Section 2, the separation between flipped bits should be at least 4 KB. This metric evaluates whether each solution meets the requirement. Solutions that satisfy this requirement are considered more practical than those that do not.

5.2. Evaluation Results

Results. Our results show that `Flip-S` consistently outperforms all baselines across all models and datasets. Specifically, Tables 1 and 2 display the results on ImageNet with 8-bit and 4-bit quantization levels, respectively. Tables 3 and 4 present the results on CIFAR-10 with 8-bit and 4-bit quantization levels. `Flip-S` achieves significantly higher ASR and CDA than the baselines, while also satisfying the BFSC requirement, whereas all other attacks fail to meet. These results underscore the superior effectiveness and practicality of `Flip-S` compared to the baselines.

For instance, on the ImageNet dataset, `Flip-S` reaches

Table 1. Results on ImageNet with 8-bit quantization level. "Clean" represents the original quantized models, and "Compromised" denotes the compromised models. `Flip-S` achieves higher CDA and ASR than all baselines across all models. Furthermore, `Flip-S` satisfies the BFSC requirement, whereas the baselines do not.

Model	Method	CDA (%)		ASR (%)	BFSC
		Clean	Compromised		
ViT-base	TBT	81.1	75.6	15.3	✗
	ProFlip	81.1	79.5	71.9	✗
	HPT	81.1	79.3	67.3	✗
	TrojViT	81.1	77.5	68.6	✗
	`Flip-S`	81.1	**80.2**	**99.5**	✓
Deit-base	TBT	83.6	70.2	31.6	✗
	ProFlip	83.6	83.1	56.8	✗
	HPT	83.6	81.3	51.5	✗
	TrojViT	83.6	82.8	79.2	✗
	`Flip-S`	83.6	**83.4**	**99.8**	✓
Deit-small	TBT	81.6	74.0	49.4	✗
	ProFlip	81.6	81.4	94.6	✗
	HPT	81.6	78.2	69.8	✗
	TrojViT	81.6	79.5	66.0	✗
	`Flip-S`	81.6	**81.5**	**99.9**	✓
SwinT-base	TBT	85.5	78.0	76.0	✗
	ProFlip	85.5	84.8	88.7	✗
	HPT	85.5	83.7	78.6	✗
	TrojViT	85.5	84.6	90.1	✗
	`Flip-S`	85.5	**84.9**	**95.2**	✓
SwinT-small	TBT	83.4	74.9	71.8	✗
	ProFlip	83.4	82.7	80.5	✗
	HPT	83.4	82.1	70.2	✗
	TrojViT	83.4	82.4	77.8	✗
	`Flip-S`	83.4	**83.1**	**98.4**	✓

Table 2. Results on ImageNet with 4-bit quantization level.

Model	Method	CDA (%)		ASR (%)	BFSC
		Clean	Compromised		
ViT-base	TBT	81.1	75.6	17.7	✗
	ProFlip	81.1	80.6	70.2	✗
	HPT	81.1	79.8	50.4	✗
	TrojViT	81.1	77.8	51.5	✗
	`Flip-S`	81.1	**80.8**	**92.4**	✓
Deit-base	TBT	83.6	71.6	50.6	✗
	ProFlip	83.4	83.3	52.3	✗
	HPT	83.4	81.5	49.5	✗
	TrojViT	83.4	82.6	48.5	✗
	`Flip-S`	83.4	**83.3**	**98.1**	✓
Deit-small	TBT	80.4	71.9	74.5	✗
	ProFlip	80.4	80.3	80.1	✗
	HPT	80.4	80.0	68.0	✗
	TrojViT	80.4	79.4	67.7	✗
	`Flip-S`	80.4	**80.3**	**98.2**	✓
SwinT-base	TBT	84.7	79.2	62.6	✗
	ProFlip	84.7	84.1	84.4	✗
	HPT	84.7	82.7	78.2	✗
	TrojViT	84.7	84.2	82.2	✗
	`Flip-S`	84.7	**84.3**	**99.5**	✓
SwinT-small	TBT	83.1	74.3	62.0	✗
	ProFlip	83.1	82.2	75.3	✗
	HPT	83.1	82.1	68.4	✗
	TrojViT	83.1	82.3	74.4	✗
	`Flip-S`	83.1	**83.0**	**99.0**	✓

Table 3. Results on CIFAR-10 with 8-bit quantization level.

Model	Method	CDA (%)		ASR (%)	BFSC
		Clean	Compromised		
ViT-base	TBT	98.6	93.4	24.3	✗
	ProFlip	98.6	98.6	76.4	✗
	HPT	98.6	96.3	54.7	✗
	TrojViT	98.6	88.2	47.1	✗
	Flip-S	98.6	**98.6**	**98.7**	✓
Deit-base	TBT	98.3	87.2	44.0	✗
	ProFlip	98.3	98.1	31.8	✗
	HPT	98.3	95.2	36.1	✗
	TrojViT	98.3	96.9	72.8	✗
	Flip-S	98.3	**98.2**	**97.7**	✓
Deit-small	TBT	97.6	95.7	83.5	✗
	ProFlip	97.6	97.1	89.4	✗
	HPT	97.6	96.7	82.2	✗
	TrojViT	97.6	96.9	78.3	✗
	Flip-S	97.6	**97.3**	**99.9**	✓
SwinT-base	TBT	98.7	75.3	32.3	✗
	ProFlip	98.7	97.6	72.5	✗
	HPT	98.7	88.5	43.8	✗
	TrojViT	98.7	85.4	42.7	✗
	Flip-S	98.7	**98.2**	**95.5**	✓
SwinT-small	TBT	97.9	74.9	52.0	✗
	ProFlip	97.9	97.7	44.5	✗
	HPT	97.9	90.2	42.7	✗
	TrojViT	97.9	87.8	60.5	✗
	Flip-S	97.9	**97.7**	**94.3**	✓

Table 4. Results on CIFAR-10 with 4-bit quantization level.

Model	Method	CDA (%)		ASR (%)	BFSC
		Clean	Compromised		
ViT-base	TBT	98.6	92.8	27.0	✗
	ProFlip	98.6	98.6	95.5	✗
	HPT	98.6	95.8	63.5	✗
	TrojViT	98.6	93.1	71.5	✗
	Flip-S	98.6	**98.6**	**99.5**	✓
Deit-base	TBT	98.1	93.1	32.0	✗
	ProFlip	98.1	97.9	58.5	✗
	HPT	98.1	96.3	42.3	✗
	TrojViT	98.1	91.4	62.7	✗
	Flip-S	98.1	**98.0**	**99.3**	✓
Deit-small	TBT	97.4	89.5	75.3	✗
	ProFlip	97.4	96.9	82.6	✗
	HPT	97.4	94.2	79.8	✗
	TrojViT	97.4	96.7	69.6	✗
	Flip-S	97.4	**97.1**	**99.5**	✓
SwinT-base	TBT	98.6	76.4	34.9	✗
	ProFlip	98.6	97.8	90.5	✗
	HPT	98.6	89.7	35.3	✗
	TrojViT	98.6	72.3	37.9	✗
	Flip-S	98.6	**98.3**	**97.3**	✓
SwinT-small	TBT	97.8	71.9	39.0	✗
	ProFlip	97.8	97.0	65.9	✗
	HPT	97.8	88.4	46.9	✗
	TrojViT	97.8	82.9	58.1	✗
	Flip-S	97.8	**97.1**	**92.1**	✓

an ASR of at least 95.2% across all models at 8-bit quantization level, whereas baselines typically achieve ASR below 80.0%. Even under the more challenging 4-bit quantization level, Flip-S maintains an ASR above 92.4% on all models, with the highest ASR among baselines being only 84.4%. In addition, Flip-S slightly degrades CDA and consistently achieves higher CDA than all baselines. For example, on the SwinT-base model at 4-bit quantization level on ImageNet, Flip-S slightly reduces CDA from 84.7% (Clean) to 84.3% (Compromised), while baselines yield CDA ranging from 79.2% to 84.2%. Furthermore, Flip-S meets the 4KB bit-flip separation compliance requirement across all models, which none of the baseline methods satisfy. This indicates that only Flip-S ensures a bit-flip separation of greater than 4 KB, underscoring its better feasibility.

Evaluation with more bits flipped. Here we do not restrict n_{max} to 50 but consider larger n_{max} to compare Flip-S with baselines. We take ImageNet dataset and Deit-base model at 4-bit quantization level as an example to evaluate when n_{max} increases to up to 500 for all BFAs. Figure 3 shows that most baselines cannot match Flip-S 's ASR even with 500 bits flipped. Flip-S achieves an ASR of 91.2% with only 25 bits flipped, while baselines require 200 to 500 bits (8×-20×) to reach comparable ASR levels. Furthermore, the best-performing baseline, ProFlip, requires 300 bits to match Flip-S's ASR with only 50 bits. These results validate our attack effectiveness.

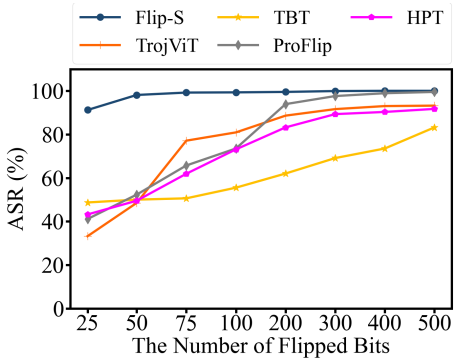

Figure 3. Comparison between Flip-S and baselines when more bits are flipped. The best-performing baseline (ProFlip) requires flipping 300 bits to achieve a similar ASR as Flip-S with only 50 bits flipped.

5.3. Ablation Study

Effects of bit-flip in scale factors. We verify the function of bit-flip within scale factors on attack effectiveness. We consider two scenarios: (1) using only triggers without any bit-flip, denoted as **Trigger-only**, and (2) flipping bits only within weights, excluding scale factors, denoted as **Weight-only**. We use the ImageNet dataset with 4-bit quantization level as an example. Table 5 shows that the ASR of Trigger-only and Weight-only scenarios is significantly lower than that of Flip-S. This indicates that these two settings cannot effectively attack ViTs, validating the importance of bit-flip within scale factors for attack effectiveness.

Table 5. Evaluation of bit-flip effects within scale factors. Trigger-only and Weight-only cannot effectively attack models, validating the importance of bit-flip within scale factors.

Method	ASR (%)				
	ViT-base	Deit-base	Deit-small	SwinT-base	SwinT-small
Trigger-only	37.0	32.2	38.3	32.1	37.6
Weight-only	68.8	81.0	88.7	96.4	93.7
`Flip-S`	**92.4**	**98.1**	**98.2**	**99.5**	**99.0**

Trigger area. We further consider varying the percentage of the trigger area (TAP) in an input image. The smaller the TAP is, the better stealthiness it has. We use the ImageNet dataset and Deit-small model with 4-bit quantization level as an example. Table 6 shows that even as TAP decreases, the ASR for `Flip-S` remains relatively high and consistently outperforms baselines with a TAP of 4.59% (see Table 2). For instance, even with TAP reduced to 1.53%, `Flip-S` achieves an ASR of 95.4%, surpassing the best baseline, ProFlip, which only reaches an ASR of 80.1% with a TAP of 4.59%. In summary, `Flip-S` can achieve better stealthiness and effectiveness than baselines.

Table 6. Impact of trigger size variation.

TAP (%)	CDA (%)		ASR (%)
	Clean	Compromised	
1.53	80.4	79.8	95.4
2.55	80.4	79.9	95.5
3.57	80.4	80.2	97.4
4.59	80.4	**80.3**	**98.2**

5.4. Potential Defense

Many defense methods target backdoor attacks [6, 11, 13, 21, 32, 35, 37, 41]. However, the literature [28, 36, 44] reveals their limitations against BFAs, which occur at inference time, bypassing traditional defenses applied during training or in the supply chain. BFAs are activated at the attacker's discretion, making continuous runtime defenses (*e.g.*, fine-pruning or clustering) resource-intensive [28]. Additionally, integrity-based defenses like Error-Correcting Code remain vulnerable as RowHammer attacks evolve to bypass these checks [8].

Several defense solutions have been proposed to mitigate bit-flip attacks. Some focus on mitigating untargeted attacks [16, 20, 24, 30, 42], which are proven to be ineffective against targeted BFAs [36]. Wang *et al.* [36] propose a solution to mitigate targeted BFAs. However, this approach is specific to CNNs and is hard to be deployed on ViTs. Özdenizci *et al.* introduce OCM [25] to mitigate targeted BFAs, and we evaluate our `Flip-S` under this defense. Furthermore, we try to mitigate attacks by preventing `Flip-S` from flipping the most critical bits. We set the number of prohibited bits to 50, 100, and 150, and observe the impact on `Flip-S` 's performance. Experiments use ImageNet dataset with 4-bit quantization level.

Table 7. Defense performance against `Flip-S`.

Defense Solutions	ASR (%)				
	ViT-base	Deit-base	Deit-small	SwinT-base	SwinT-small
OCM	91.6	96.4	93.3	98.5	97.8
50 prohibited bits	76.1	88.3	92.8	95.1	95.2
100 prohibited bits	69.4	79.1	88.2	90.1	91.9
150 prohibited bits	71.0	75.0	91.8	94.8	90.6
`Flip-S` (no defense)	92.4	98.1	98.2	99.5	99.0

Table 7 shows that OCM cannot effectively mitigate `Flip-S`. The main reason is that OCM protects the last layer, whereas `Flip-S` can flip bits in any layer. Preventing critical bit-flips improves resilience compared to OCM, yet it still cannot fully defend against `Flip-S`. For example, even when prohibiting 150 critical bits, the ASR on DeiT-small, SwinT-base, and SwinT-small remains above 90.0%. Furthermore, even if defenders identify and restrict critical bits, any changes in the trigger would shift these critical bits, thereby compromising the defense.

6. Conclusion

We introduced `Flip-S`, a novel, practical targeted BFA against ViTs. This work is the first to highlight the significance of bit-flips in scale factors, introducing a `SFS` to identify critical bits in these factors. Furthermore, we implement a mutual exclusion strategy to maintain a 4 KB minimum separation between flipped bits, ensuring attack feasibility. Extensive experiments on two datasets across five ViT architectures and two quantization levels demonstrate that `Flip-S` outperforms existing attacks in both effectiveness and practicality.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. This research is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative, National Natural Science Foundation of China (No. U24A20337, No. 62172027, No. U24B20117), and Zhejiang Provincial Natural Science Foundation of China (No. LZ23F020013). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] bitsandbytes - the bitsandbytes library is a lightweight python wrapper around cuda custom functions, in particular 8-bit optimizers, matrix multiplication, and 8 & 4-bit quantization functions. https://github.com/

bitsandbytes-foundation/bitsandbytes. Accessed Jan, 2024. 6

[2] Samira Abnar and Willem H. Zuidema. Quantifying attention flow in transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, 2020. 3, 4

[3] 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, CVPR*, 2024. 1, 2, 3

[4] 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 *9th International Conference on Learning Representations, ICLR*, 2021. 1, 2

[5] 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 *Computer Vision - ECCV European Conference*, 2022. 1, 2, 3, 6

[6] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019. 8

[7] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV*, 2021. 1, 2, 3, 6

[8] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ECC memory against rowhammer attacks. In *IEEE SP*, 2019. 8

[9] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2022. 1, 2

[10] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2023. 1, 2

[11] Khoa D. Doan, Yingjie Lao, Peng Yang, and Ping Li. Defending backdoor attacks on vision transformer via patch processing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI*, 2023. 8

[12] 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 *9th International Conference on Learning Representations, ICLR*, 2021. 1

[13] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. STRIP: a defence against trojan attacks on deep neural networks. In *ACSAC*, 2019. 8

[14] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *IEEE/CVF International Conference on Computer Vision, ICCV*, 2019. 1, 2

[15] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. MCDNN: an approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys*, 2016. 2

[16] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *CVPR*, 2020. 8

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018. 1, 2

[18] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: scalable rowhammering in the frequency domain. In *43rd IEEE Symposium on Security and Privacy, SP*, 2022. 1, 3, 6

[19] 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 41st International Symposium on Computer Architecture, ISCA*, 2014. 1

[20] Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Defending bit-flip attack through DNN weight reconstruction. In *ACM/IEEE DAC*, 2020. 8

[21] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *ICLR*, 2021. 8

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE/CVF International Conference on Computer Vision, ICCV*, 2021. 1, 5

[23] Onur Mutlu and Jeremie S. Kim. Rowhammer: A retrospective. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2020. 1

[24] Najmeh Nazari, Hosein Mohammadi Makrani, Chongzhou Fang, Hossein Sayadi, Setareh Rafatirad, Khaled N. Khasawneh, and Houman Homayoun. Forget and rewire: Enhancing the resilience of transformer-based models against bit-flip attacks. In *33rd USENIX Security Symposium, USENIX Security*, 2024. 1, 8

[25] Ozan Özdenizci and Robert Legenstein. Improving robustness against stealthy weight bit-flip attacks by output code matching. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2022. 8

[26] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE EuroS&P*, 2016. 2

[27] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, 2019. 1, 2, 3

[28] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. TBT: targeted neural network attack with bit trojan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020. 1, 2, 3, 6, 8

[29] 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*, 2021. 1

[30] Adnan Siraj Rakin, Li Yang, Jingtao Li, Fan Yao, Chaitali Chakrabarti, Yu Cao, Jae-sun Seo, and Deliang Fan. RA-BNN: constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy. *CoRR*, abs/2103.13813, 2021. 8

[31] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-BFA: targeted bit-flip adversarial weight attack. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2022. 1

[32] Akshayvarun Subramanya, Aniruddha Saha, Soroush Abbasi Koohpayegani, Ajinkya Tejankar, and Hamed Pirsiavash. Backdoor attacks on vision transformers. *CoRR*, abs/2206.08477, 2022. 8

[33] M. Caner Tol, Saad Islam, Andrew J. Adiletta, Berk Sunar, and Ziming Zhang. Don't knock! rowhammer at the backdoor of DNN models. In *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Network, DSN*, 2023. 1, 2, 3

[34] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021. 5

[35] 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, SP*, 2019. 8

[36] Jialai Wang, Ziyuan Zhang, Meiqi Wang, Han Qiu, Tianwei Zhang, Qi Li, Zongpeng Li, Tao Wei, and Chao Zhang. Aegis: Mitigating targeted bit-flip attacks against deep neural networks. In *32nd USENIX Security Symposium, USENIX Security*, 2023. 1, 3, 6, 8

[37] 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 *ECCV*, 2020. 8

[38] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *CoRR*, abs/2006.03677, 2020. 2, 5

[39] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium, USENIX Security*, 2020. 1, 2, 3

[40] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2022. 1, 2

[41] Yi Zeng, Si Chen, Won Park, Zhuoqing Mao, Ming Jin, and Ruoxi Jia. Adversarial unlearning of backdoors via implicit hypergradient. In *ICLR*, 2022. 8

[42] Jinyu Zhan, Ruoxu Sun, Wei Jiang, Yucheng Jiang, Xunzhao Yin, and Cheng Zhuo. Improving fault tolerance for reliable dnn using boundary-aware activation. *TCAD*, 2021. 8

[43] Zhaoyang Zhang, Wenqi Shao, Jinwei Gu, Xiaogang Wang, and Ping Luo. Differentiable dynamic quantization with mixed precision and adaptive resolution. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021. 1, 2

[44] Mengxin Zheng, Qian Lou, and Lei Jiang. Trojvit: Trojan insertion in vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2023. 1, 2, 3, 6, 8

[45] Xuan Zhou, Souvik Kundu, and Peter Anthony Beerel. What makes vision transformers robust towards bit-flip attack? In *ICLR Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. 1, 2