Aegis: Mitigating Targeted Bit-flip Attacks against Deep Neural Networks

Jialai Wang[*], Ziyuan Zhang[†], Meiqi Wang[*], Han Qiu[*§✉], Tianwei Zhang[‡], Qi Li[*§], Zongpeng Li[*♣✉], Tao Wei[♠], and Chao Zhang[*§]

[*]*Tsinghua University,* [†]*Beijing University of Posts and Telecommunications,* [‡]*Nanyang Technological University,* [♣]*Hangzhou Dianzi University,* [♠]*Ant Group,* [§]*Zhongguancun Laboratory*

{wang-jl22, wang-mq22}@mails.tsinghua.edu.cn, zhangziy0421@bupt.edu.cn, {qiuhan, qli01, zongpeng, chaoz}@tsinghua.edu.cn, tianwei.zhang@ntu.edu.sg, lenx.wei@antgroup.com

Abstract

Bit-flip attacks (BFAs) have attracted substantial attention recently, in which an adversary could tamper with a small number of model parameter bits to break the integrity of DNNs. To mitigate such threats, a batch of defense methods are proposed, focusing on the untargeted scenarios. Unfortunately, they either require extra trustworthy applications or make models more vulnerable to targeted BFAs. Countermeasures against targeted BFAs, stealthier and more purposeful by nature, are far from well established.

In this work, we propose `Aegis`, a novel defense method to mitigate targeted BFAs. The core observation is that existing targeted attacks focus on flipping critical bits in certain important layers. Thus, we design a dynamic-exit mechanism to attach extra internal classifiers (ICs) to hidden layers. This mechanism enables input samples to early-exit from different layers, which effectively upsets the adversary's attack plans. Moreover, the dynamic-exit mechanism randomly selects ICs for predictions during each inference to significantly increase the attack cost for the adaptive attacks where all defense mechanisms are transparent to the adversary. We further propose a robustness training strategy to adapt ICs to the attack scenarios by simulating BFAs during the IC training phase, to increase model robustness. Extensive evaluations over four well-known datasets and two popular DNN structures reveal that `Aegis` could effectively mitigate different state-of-the-art targeted attacks, reducing attack success rate by 5-10×, significantly outperforming existing defense methods. We open source the code of `Aegis`[1].

1 Introduction

The recent revolutionary development of deep neural network (DNN) models has promoted various security- and safety-sensitive intelligent applications, such as autonomous driving [52], AI on satellites [13], and medical diagnostics [55]. An adversary could manipulate data used by DNN models and model parameters to launch various attacks. The security and robustness of DNN models have become the key factors affecting the deployment of these systems. A significant amount of research efforts have been devoted to protecting DNN models from data-oriented attacks, e.g. adversarial attacks [7, 14, 23, 40, 41, 57] that manipulate inference data, or DNN backdoor attacks [2, 11, 33, 35, 37, 39, 53, 68] that manipulate training data. These efforts can secure the model against data-oriented threats. But little attention has been paid to mitigate the emerging parameter-oriented attacks.

Recent studies have shown that well-trained DNN models are vulnerable to parameter-oriented attacks, which tamper with model parameters [8, 45–48, 64]. For instance, flipping a small number of critical bits (i.e. $0 \rightarrow 1$ or $1 \rightarrow 0$) of off-the-shelf DNN model parameters can trigger catastrophic changes in the inference process [20, 45], lowering the prediction accuracy or manipulating the inference to any target labels. These bit-flip attacks (BFAs) experiment in real-world scenarios. DeepHammer [64] performs BFAs on a PC via rowhammer. Also, BFAs are performed on the multi-tenant FPGA devices in cloud-based machine learning services [48].

Current state-of-the-art BFAs can be classified into untargeted and targeted attacks. The *untargeted* BFAs aim to comprise the victim model accuracy to the random guess level [45, 48, 64]. For instance, with optimized critical bit search algorithms, the BFA in [45] needs to flip only 13 out of 93 million bits of an 8-bit quantized ResNet-18 model on ImageNet, to degrade its top-1 accuracy from 69.8% to 0.1%. In comparison, the *targeted* BFAs are stealthier, by misleading target models to target labels on specific samples (i.e., sample-wise attacks) or samples with special triggers (i.e., backdoor attacks) while preserving model accuracy for other samples. For instance, the sample-wise BFA [4] can flip only less than 8 critical bits on average of an 8-bit quantized ResNet-18 model on ImageNet to manipulate the prediction of specific input samples.

Existing defenses for data-oriented attacks, e.g., adversarial training, are proven not useful to mitigate BFAs [19]. Instead,

✉ Corresponding authors.

[1]https://github.com/vul337/Aegis.git

a small number of dedicated defenses have been proposed to mitigate BFAs, which can be classified into two categories, namely integrity verification, and model enhancement.

First, integrity verification-based approaches [25, 36, 38] verify the integrity of model parameters at runtime to detect BFAs. For instance, HashTAG [25] verifies the integrity of model parameters on the fly by extracting and comparing the unique signatures of the original and the runtime DNN models. A low-collision hashing scheme could be used for generating signatures and achieving almost zero false positives. This type of approach could detect any attempts to tamper with the models and thus could defeat both targeted and untargeted BFAs. However, they in general require an additional *trusted* and *secure* monitoring process to *continuously* monitor the target DNN model. It introduces additional performance overhead and resource cost and is not applicable to commercial off-the-shelf devices and practical scenarios.

Second, model enhancement-based approaches [32, 67] focus on improving the robustness of target models directly, making BFAs difficult or impossible to launch. Note that, precisely flipping a large number of bits via hardware-level attack is not practical [64]. An important metric for evaluating the difficulty or cost of BFAs is *the number of bits to flip* (e.g. DeepHammer [64] sets 24 bits as the maximum number of bits can be flipped). Thus, this approach aims to significantly increase the number of bits to flip for achieving the same attack goals. One promising approach is to quantize the model parameters to constrain the weights' value range that may potentially be changed by BFAs. For instance, a binarization architecture, BNN, is proposed in [19] to retrain the model from scratch to generate a model with weights only equal to -1 or +1. An enhanced version, RA-BNN [49] further extends the quantization on the activation function outputs by -1 or +1. This approach can significantly increase the difficulty for untargeted attacks (e.g. flipping $40 \sim 300\times$ more bits in [19], which is infeasible). However, this solution can only mitigate untargeted attacks, while making the model even more vulnerable to targeted attacks. According to our experiments, TBT attacks [46] can achieve a similar success rate by flipping even fewer bits in a binarization-trained model (from 206 bits in the vanilla model to 50 bits in the corresponding binarization model. See details in Section 5.3).

In this work, we propose `Aegis`[2], a novel method to mitigate different targeted BFAs. Our key observation is that existing targeted BFAs achieve their goals by locating the most critical bits according to the model inference process. Particularly, they flip the critical bits in either the final layer of the target model or the most important layer determined by some optimization methods. Based on the observation, we design our solution by using a *dynamic multi-exit* architecture to train extra internal classifiers (ICs) for hidden layers [27]. These ICs can distribute the early exit of input samples to different hidden layers of the target model. This can mitigate the existing attacks which flip bits in one specific layer. Furthermore, considering the adaptive BFAs where the defense is transparent, adversaries could use the sample exit distribution to locate critical hidden layers to flip their critical bits. `Aegis` can mitigate such adaptive BFAs by dynamic masking certain exits. Lastly, we design robust training on ICs to simulate the attacks when the critical bits in each hidden layer are flipped. This can defeat more sophisticated adaptive BFAs that include all exits to flip critical bits in all layers of the model.

[2] `Aegis` is a powerful shield carried by Athena and Zeus in Greek mythology, which can defeat various attacks from thunders, hammers, etc.

`Aegis` aims to achieve three important goals, i.e. *non-intrusive*, *platform-independent*, and *utility-preserving*. It can protect the model without modifying any of its parameters. This will exclude the defenses that retrain the model from scratch, which is either too costly on a large-scale dataset or infeasible when training datasets are unavailable. We design `Aegis` at the application level without requiring any additional reliable program or hardware protection, making it generally applicable. For utility-preserving, `Aegis` has negligible impacts on the prediction accuracy.

We conduct extensive experiments to evaluate `Aegis` against three state-of-the-art targeted BFAs with different goals, as well as their potential adaptive attack counterparts. We consider two well-known model architectures (ResNet-32 and VGG-16) and four datasets (CIFAR-10, CIFAR-100, STL-10, and Tiny-ImageNet). The results show that we can mitigate different BFAs by significantly increasing the number of bits to flip (e.g. flip $35\times$ more bits to achieve a similar attack success rate of [8]) or reducing their attack success rate to a low level (e.g. keep attack success rate lower than 4% with a similar number of bits flipped as [4]).

2 Background

2.1 Targeted Bit-flip Attacks

We introduce three state-of-the-art targeted BFAs, i.e., TBT attack [46], ProFlip attack [8] and TA-LBF attack [4].

TBT [46] is a targeted BFA that injects backdoors into the target model through flipping bits. The attacker's goal is that the compromised model still operates with normal inference accuracy on benign inputs but makes mistakes on samples with specific triggers. Specifically, when the adversary embeds the trigger into any input, the model is forced to classify this input to a certain target class. Note that this method only flips bits in the final layer of the target model. In the final layer, the adversary first selects w_b critical network neurons which have the most significant impact on the target class, then generates a specific trigger to activate these neurons. Finally, the adversary formalizes an optimization problem to modify critical bits corresponding to these neurons.

ProFlip [8] inserts a backdoor into the target model by flipping bits in the network weights to manipulate the prediction of all inputs attached with the trigger to a certain target class.

This method could flip bits in all the layers of the model by selecting salient neurons through forwarding derivative-based saliency map construction (also known as jacobian saliency map attack (JSMA) [43]). Then the adversary uses the gradient descent method to generate triggers, which can stimulate salient neurons to large values. Finally, ProFlip proposes an efficient retrieval algorithm to select the optimal parameter, and determine critical bits in the parameter to flip.

TA-LBF [4] does not need a trigger but only misclassifies a specific sample to a target class by flipping the critical bits of the parameters, which makes the attack stealthier than TBT and ProFlip. The adversary formalizes the attack as binary integer programming since the parameters are stored as binary bits (i.e., 0 and 1) in the memory. It further equivalently reformulates this binary integer programming problem as a continuous optimization problem. Using the alternating direction method of multipliers (ADMM) method [63] solves the optimization problem to determine critical bits to flip.

2.2 Existing Defense and Analysis

Existing defense methods could be categorized into two types. Details are given as follows.

Model enhancement. `Aegis` also falls into this defense category. Li et al. [32] adopt a weight reconstruction method, which could defuse the changed values on several parameters to multiple parameters, thus mitigating the effects brought by untargeted BFAs. Zhan et al. [67] modify rectified linear unit (ReLU), a commonly used activation function in DNNs, to tolerate the faults incurred by bit-flipping on weights.

The above two defense methods are proved to be less effective than the binarization strategy such as BIN [19] and RA-BNN [49]. This strategy applies the binarization-aware training [50] to retrain a binarization model from scratch to mitigate untargeted BFAs. Its point is to constrain the range of parameters' values to force attackers to flip more bits in order to achieve the same attack success rate. Specifically, BIN [19] converts a part of the model parameters from high precision, e.g., 32-bit floating-point, to a binary format ($\{-1,+1\}$). RA-BNN [49] uses a more aggressive way to further quantize the output of activation functions to $\{-1,+1\}$ as well. Although these methods can effectively mitigate untargeted attacks, they still have three limitations. First, they require retraining a target model from scratch, which introduces significant computation costs. Second, aggressive precision reduction on models will affect model accuracy. Third, more importantly, they make the model even more vulnerable to targeted attacks such as TBT [46] (See Section 5.3).

Integrity verification. This approach is orthogonal to model enhancement, which protects models from another dimension. One approach [15, 25, 31, 38] is to apply the integrity verification to defend BFAs is that the defender extracts a ground-truth signature from the model before deployment. Once the model is deployed, new hashes are extracted during inference to compare with the ground-truth one.

This approach can also be realized at the hardware or system level based on techniques such as ECC [10]. However, it has three main practical obstacles. (1) They are restricted to specific platforms, e.g., [15] requires new CUDA kernel for integrity protection and [26] requires new processors with targeted row refresh. Also, some techniques such as ECC are not deployed in some embedded devices such as Nvidia Nano or Jetson AGX Xavier. (2) These methods are not absolutely secure against bit-flip attack [10]. (3) They only detect whether a model is changed in memory, but do not provide mitigation against specific attacks.

Comparison with existing defense methods. Existing model enhancement methods could effectively mitigate untargeted BFAs, but pay no attention to targeted BFAs. Compared with untargeted attacks, targeted attacks are more threatening and stealthier, as the compromised model could still behave normally on clean samples. Thus, we aim to fill this gap. Besides, `Aegis` is non-intrusive compared with existing methods as we do not modify the original models or retrain from scratch.

Integrity verification approaches experiments via hardware or system-level solutions. Instead, `Aegis` aims to give an application-level solution that is generally effective regardless of the underlying hardware circuits, operating systems, or DL libraries. Besides, `Aegis` is orthogonal to integrity verification such as ECC so `Aegis` can provide extra protection on ECC-enabled systems on different levels.

We also notice there are defense methods that are specific to DNN backdoor attacks [12, 34, 61, 66]. However, they have different threat models with mitigating BFAs. Specifically, they aim to detect or remove an existing backdoor in offline trojan models. However, BFAs usually experiment on a deployed clean model under attack at runtime.

2.3 Multi-exit DNN models

The initial motivation for setting exits for inference at hidden layers is to solve the overthinking issue. Since the growing performance of modern DNN models brings a significantly increasing number of layers and parameters in most of the state-of-the-art DNN models, Huang *et al.* [22] point out that forcing all samples, especially canonical samples to inference all layers of a DNN model definitely brings a waste of energy and time. Moreover, Kaya *et al.* [27] find that forcing certain samples classified correctly with only a few shallow layers but inference through all layers will lead to the wrong prediction.

Many solutions have been proposed to let samples early exit the model to address the above issues [17, 18, 22, 62]. One promising technique is the shallow-deep network (SDN) [27]. The key insight of SDN is that during the inference process for a sample, it is highly possible that some layer in the middle of the network already has high confidence for prediction. So it can early exit from the model without the need to go through all the layers to significantly reduce the inference time and energy consumption. It is very convenient to convert a vanilla DNN model (e.g. ResNet) into an SDN model. We can select some appropriate convolution layers, and attach an internal classifier (IC) to each of them to form an early exit. When the prediction confidence of the input sample as one label is higher than a threshold at an exit, the inference will stop and output that label. A proper threshold can realize early-exit with a tiny accuracy loss.

Deploying multi-exit model architectures such as SDN for security purposes is proposed in [21, 69] to mitigate adversarial attacks. Particularly, an input-adaptive multi-exit DNN structure with a dynamic inference process can mitigate the adversarial perturbation generation and further increase the difficulty of adaptive adversarial attacks. However, since BFAs aim at manipulating model parameters rather than input samples, simply deploying multi-exit DNN structures cannot achieve the defense requirements and more sophisticated methods are needed.

3 Threat Model and Defense Requirements

We consider an adversarial scenario, where the adversary is able to perform BFAs against the victim DNN models. He can precisely flip a number of parameter bits to affect the model prediction results. The exploitability and practicality of such threats have been validated and evaluated in previous works [6, 44, 48, 64]. For instance, attackers verified an untargeted BFA on DNNs with a PC platform via row-hammer [48]. Also, the adversary can co-locate his malicious program on the same machine with the victim DNN model and then use methods like row-hammer [48] to perform BFAs. The feasibility of BFAs is also validated in our paper on a PC platform following previous works [44, 64] (see details in Section 5.7).

Following the previous works [4, 8, 44, 46, 64], we assume the adversary has very strong capabilities. He has full knowledge of the victim model, including the DNN architecture, model parameters, etc. We further assume that the adversary knows every detail of any possible defense deployed in the system, such as the mechanism, algorithm, parameters, etc. If a defense solution employs randomization-based techniques, we assume the random numbers generated in real-time are perfect with a large entropy such that the adversary cannot obtain or guess the correct values. It is worth noting that these assumptions represent the strongest adversary, which significantly increases the difficulty of defense designs.

Adversarial goals. Previous works have demonstrated different goals for BFAs, as summarized below:

- **Untargeted attack**. The adversary aims to drastically degrade the overall accuracy of the victim model. A powerful untargeted attack can decrease the model accuracy to nearly random guess after the exploitation [64].
- **Backdoor targeted attack**. The adversary designs a specific trigger and injects the corresponding backdoor [8] into the DNN via the BFAs. Then for any input sample containing the trigger, the compromised model will mispredict it as the target class.
- **Sample-wise targeted attack**. The adversary aims to tamper with the model such that it only mispredicts a specific sample as the target class while having normal predictions for other samples.

In this paper, we focus on the last two categories of targeted BFAs due to two reasons. (1) As indicated in Section 2.2, a number of past works have explored the mitigation approaches against untargeted BFAs. In contrast, how to effectively thwart targeted BFAs is rarely investigated. (2) The backdoor or sample-wise targeted BFAs are much stealthier than the untargeted attack, as the compromised model behaves normally for clean samples. This significantly increases the defense difficulty and an effective solution is urgently needed. Besides, although untargeted BFAs are not within our scope, our approach can mitigate untargeted BFAs (see Section 6).

Defense requirements. The purpose of this paper is to design a defense approach to comprehensively protect DNN models from different targeted BFAs and their possible adaptive attacks. It is worth highlighting that our goal is to increase the attack cost rather than totally preventing BFAs. Theoretically, the adversary can tamper with more parameter bits even if a strong defense is applied. So we aim to significantly increase the number of flipped bits required to achieve the desired adversarial goal, thus making the attack less feasible or practical. Our defense requirements are as follows.

- **Non-intrusive**. The defense should be easy to deploy on off-the-shelf DNN models. The defender does not need to modify parameters of the original model, e.g., retraining a model with binarization [19] from scratch since this can incur significant computation cost, especially for large-scale DNN models (e.g. ImageNet scale [51]).
- **Platform-independent**. Previous works propose hardware or system-level solutions to prevent fault injection attacks, e.g., new CUDA kernels for integrity protection [15], new processors with targeted row refresh [26]. However, these solutions are restricted to some specific platforms. Instead, we hope to have an application-level solution that is generally effective regardless of the underlying hardware circuits, operating systems, and deep learning libraries.
- **Utility-preserving**. The defense solution should have a negligible impact on the model inference process. It should preserve the usability of the original model without hugely decreasing its prediction accuracy.

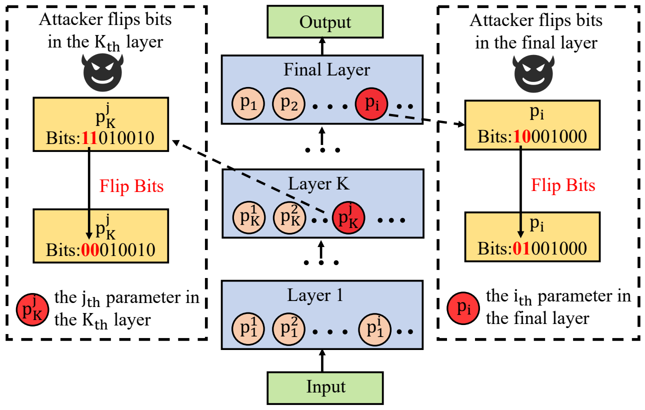
4 Methodology

4.1 Design Insight

We propose `Aegis`, a novel approach to mitigate different types of targeted BFAs. Our approach is composed of a

Dynamic-Exit SDN (DESDN) mechanism followed by a robust training (ROB) strategy only on ICs. We illustrate our design insight via three steps as follows.

First, there are BFAs that only flip bits in the final layer since the parameters of the final layer are directly related to the prediction results. It is straightforward to deduce that a multi-exit mechanism can thwart the basic BFAs that flip bits only in the final layer (as shown in Figure 1). (1) Using a multi-exit DNN structure such as the SDN can interfere with the adversarial perturbations carried by the samples (triggers generated only from the vanilla model). Malicious samples may exit early to stop inference at an arbitrary hidden layer which generates different predictions compared with the inference on the target vanilla model. (2) By forcing most samples to exit early, the flipped bits at the final layer will be probably ignored during inference to achieve the defense goals.


Attacker flips bits in the K_th layer
p_K^j
Bits:11010010
Flip Bits
p_K^j
Bits:00010010
the j_th parameter in the K_th layer
Output
Final Layer
Layer K
Layer 1
Input
Attacker flips bits in the final layer
p_i
Bits:10001000
Flip Bits
p_i
Bits:01001000
the i_th parameter in the final layer
Figure 1: An example of flipping bits in the K_{th} or final layer.

Second, we consider the existing more sophisticated BFAs that are not targeting only the final layer. For instance, the adversary may use an optimal way to locate the critical bits in the hidden layers (e.g. flipping bits in the K_{th} layer in Figure 1). Directly using the SDN structure cannot provide protection when the critical bits are flipped in the shallow layers. Moreover, in a white-box scenario, the adversary can observe the exit distribution of samples to locate critical exits for performing attacks in the corresponding critical layers (e.g. 78% samples will exit in the last five layers of a VGG-based SDN model). Thus, we propose a Dynamic-Exit SDN (DESDN) mechanism that randomizes the exit for each inference. This DESDN can mitigate the case when the adversary flips bits in shallow layers since the sample exits the model in a random layer, which has a low probability of containing the flipped bits. Moreover, DESDN can push the exit distribution to a uniform one (see our experiments in Figure 5 (a)) such that there are no critical exits for the adversary to consider.

Third, we further consider the most powerful adaptive attack. With the knowledge of all the details of Aegis, the adversary may include all exits to optimize his critical bit search. Although this will increase the attack cost (more bits to flip), it is possible to flip bits, particularly targeting at Aegis to achieve the attack regardless of where the sample exits. Our insight is to further design a robust training (ROB) strategy to find the critical and vulnerable bits for clean samples' inference process and simulate the influence when they are flipped. Note we only perform ROB on ICs without touching the target vanilla model to guarantee the non-intrusive defense requirement. This can improve the robustness of ICs to mitigate the significant convolutional output change when certain bits are flipped. Therefore, the difficulty of performing bit-flip attacks will be further increased. Below we detail two core components (DESDN and ROB) of Aegis and analyze its security against various attacks.

4.2 Dynamic-Exit SDN (DESDN)

As the first component of Aegis, DESDN consists of two steps: converting a model M to an SDN model $\hat{M}$ (offline), and performing a random exit strategy during inference (online).

4.2.1 Stage 1: Constructing SDN Model

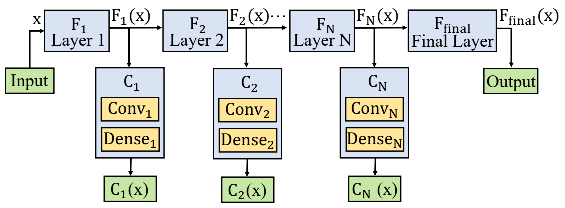


Figure 2: An example of the model attached by ICs. C_i denotes the i-th IC attached by us. During inference, each IC could make a prediction. For example, $C_i(x)$ is the prediction of C_i.

We adopt the technique in prior work [27] to build the SDN model $\hat{M}$ from M, which shows negligible accuracy degradation during conversion. Specifically, we assume M consists of N internal layers F_i, $(1 \le i \le N)$, and ends with the final layer F_{final}. For an inference sample, M performs the classification as $M(x) = F_{final}(F_N(,...F_1(x)))$. For simplicity, we denote the output of the i-th internal layer as $F_i(x)$, and the output of the final layer $M(x)$ as $F_{final}(x)$. Then our goal is to train an IC (C_i) for each internal layer i, as shown in Figure 2. Then C_i is attached to layer i and makes the prediction $C_i(F_i(x))$, which is simplified as $C_i(x)$. To restrict the size of the IC, each C_i only contains one convolutional layer and one dense layer. Such a simple structure makes it efficient to learn the parameters while maintaining high classification accuracy. This design is general and can be applied to different models.

During construction, the defender freezes the parameters of the original model M but just trains the ICs. Note, this training process is much more efficient than training a complete model from scratch. For instance, training ICs for a vanilla model is $3.2 \sim 8.2\times$ faster than training this model. Further, there is a trade-off between IC training cost and model accuracy. For instance, if we allow a tiny accuracy drop (e.g. 2% like previous work [19]), IC training cost will be less than 10% of the original model training and can be negligible. We leave further reduction of training cost as our future work.

4.2.2 Stage 2: Randomizing Exits During Inference

After attaching the trained ICs to M, the constructed SDN model $\hat{M}$ allows early exit. A threshold τ is then introduced to judge whether the inference should exit at each internal layer F_i. Specifically, for a given sample x, when the inference process reaches the i-th layer, we compute the confidence score of the corresponding IC $max(C_i(x))$. If this score is larger than τ, then the process will exit from C_i with the corresponding output without going into deeper layers. This deterministic exit mechanism can thwart the basic BFAs, but may still be vulnerable to adaptive attacks.

To further secure the inference computation, we design a dynamic exit strategy. Particularly for each query sample x, among all the ICs $C = \{C_0, C_1, ..., C_N, F_{final}\}$, we randomly select a set of q candidate ICs, denoted as $\hat{C}$. Then we perform the early exit within these candidate ICs based on their confidence scores: we find the first IC C_i in $\hat{C}$ whose confidence score $max(C_i(x))$ is larger than the threshold τ_i. Then this layer is selected as the early exit for this inference sample. If none of the candidate ICs can satisfy the early exit criteria, we will choose the final layer in $\hat{C}$ as the exit for prediction. There exists a trade-off between model accuracy and security, determined by the hyper-parameter q. Specifically, a smaller q can make the selected exit more random with larger entropy. However, it also increases the probability that these q ICs cannot meet the early exit criteria, and the prediction has lower confidence. In Section 5 we will show that we can find the appropriate value of q that brings high stochasticity to the exit selection, with negligible impact on the model accuracy.

4.3 Robust Training on ICs

We then introduce the second core component of `Aegis`: `ROB`, which further enhances the defense effectiveness against BFAs. In particular, when the adversary flips bits in the i-th layer of M, it could still possibly affect any ICs after this layer with a certain probability, as the layer output $F_i(x)$ passes to these ICs and affects their predictions. To reduce such impacts, we propose `ROB` to improve the robustness of the attached ICs. Note here the `ROB` is only performed on ICs so there is no modification on the target model.

The key insight of `ROB` is to help ICs adapt to the cases when critical bits are flipped in M. In general, without prior knowledge of the BFAs, we construct a bit-flipped model by considering only the benign samples' inference process to simulate the target compromised model. Then, we craft new samples based on this bit-flipped model for `ROB`. These training samples simulate the outputs of flipped layers, and ICs will learn such data to correct the prediction from the adversarial scenarios. Figure 3 illustrates the basic idea of this approach, which consists of two steps.

The first step is to construct the flipped model. For defense generality, we assume the defender does not know the exact attack methods to mitigate. In this case, to make the flipped model closer to the real-world victim model, we design a vulnerable-protection algorithm (VPA) to figure out vulnerable bits that could be potentially flipped. Our VPA aims to find bits that are critical to the model decisions. Such bits might significantly affect the prediction results and are vulnerable to being flipped by the adversary. We note that existing attack methods [4, 8, 45, 46, 64] treat gradients w.r.t bits as a key component to select flipped bits. Indeed, the gradients of the model output w.r.t bits reflect the importance of bits in model decisions. Inspired by this, the basic idea of our VPA is to select critical bits according to the gradients of inference loss $\mathcal{L}_{inf}$ w.r.t bits, where $\mathcal{L}_{inf}$ is defined as follows:

$$\mathcal{L}_{inf} = \mathcal{L}_{ce}(F_{final}(x); l), \tag{1}$$

where $\mathcal{L}_{ce}$ is the cross-entropy loss, l is the ground-truth label of the input sample x. Below we describe VPA in detail.

In particular, given a target model M, we denote all bits in M as B. We first establish a substitute model which is exactly the same as M. In each iteration of VPA, (1) we follow previous work [45] to calculate the gradients of $\mathcal{L}_{inf}$ w.r.t. each bit b ($b \in B$), denoted as $\nabla_b \mathcal{L}_{inf}$. (2) Then, we descendingly rank the vulnerability of bits by the absolute value of their $\nabla_b \mathcal{L}_{inf}$, and select the bits with the top-k gradients. (3) We treat these bits as vulnerable bits, and flip them. We iterate the above process until the maximum iteration budget N_{vpa} exhausts to get the flipped model. Note that in each iteration, VPA must recalculate the gradients of each bit: as the model M is dynamically changed due to the flipped bits, old gradients could not reflect the importance of bits in the newly changed model decisions. After this step, we can get the flipped model $\hat{M}$, which consists of N internal layers $\hat{F}_i$, $(1 \le i \le N)$.

The second step is to synthesize training samples for `ROB`. We freeze the original weights and only train weights in ICs. Given an input sample x from the original training set, we denote the output of the layer i from the flipped model $\hat{M}$ as $\hat{F}_i(x)$. So $\hat{F}_i(x)$ will serve as the special training sample for the IC attached to layer i. It can help the IC better adapt to the attack scenarios in advance, so as to improve the robustness of the IC. During training, except for $\hat{F}_i(x)$, we also need to apply original training data $F_i(x)$ provided by M, so as to ensure the accuracy of ICs.

4.4 Security Analysis

After introducing the design of `Aegis`, we give a comprehensive qualitative analysis of the resilience of this methodology against different types of BFAs. The first two are basic attacks from existing works, while the last one depicts a possible adaptive strategy targeting our new defense mechanism.

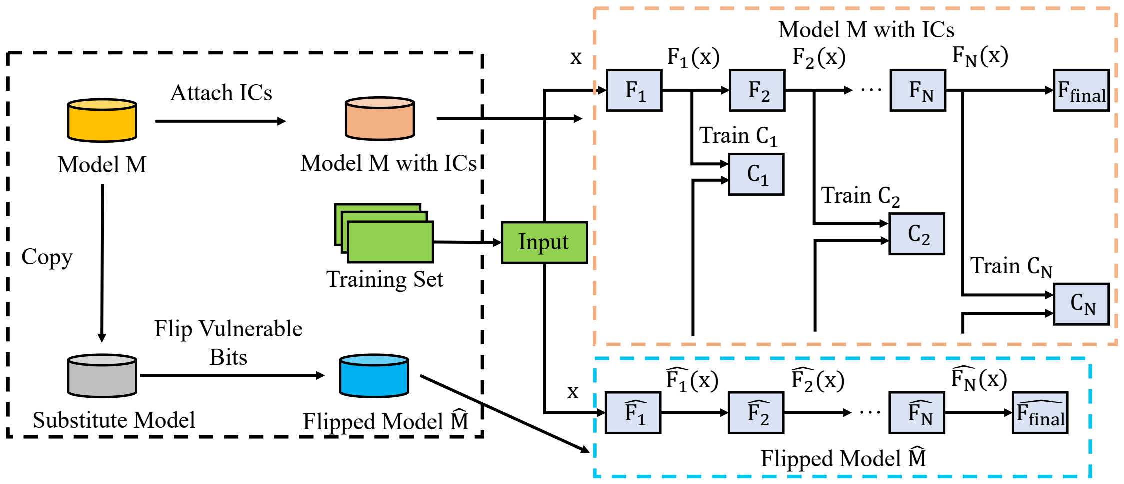


Figure 3: Illustration of ROB. We first construct the flipped model $\hat{M}$, and then synthesize training samples for ROB. In particular, we use $\hat{M}$ to generate special training data, e.g., $\hat{F}_1(x)$. Then, we apply these special training data, together with the original training data to train ICs. For example, we use $F_1(x)$ and $\hat{F}_1(x)$ to train C_1.

Attack I: the adversary only flips the bits in the final layer of the model. This strategy is adopted in TBT [46] and TA-LBF [4]. The basic SDN model can defeat these two attacks.

Formally, the TBT attack defines a target class t and a specific trigger Δ for activating the backdoor in the victim model. Given a clean input sample x with the ground truth label l, the attacker aims to make the compromised model mispredict $x+\Delta$ as t. Identification of the critical bits can be modeled as an optimization problem, which aims to minimize the following loss function:

$$\mathcal{L}_{TBT} = \mathcal{L}_{ce}(F_{final}(x);l) + \mathcal{L}_{ce}(F_{final}(x+\Delta);t), \quad (2)$$

where $\mathcal{L}_{ce}$ is the Cross-Entropy loss function. We observe that the loss function only considers the final layer. As a result, the basic SDN model is able to thwart this attack, as most inference samples will exit earlier before the final layer F_{final}, and not be affected by the flipped bits.

Similarly, TA-LBF is a sample-wise attack, which aims to cause the misclassification of a specific sample x from its ground truth label l to the target label t. This can also be formulated as an optimization problem:

$$\mathcal{L}_{TA-LBF} = \mathcal{L}_1(x;l;t;F_{final}) + \lambda\mathcal{L}_2(x;l;t;F_{final}), \quad (3)$$

where λ is a hyperparameter, $\mathcal{L}_1$ and $\mathcal{L}_2$ are two specific loss functions that ensure the attack effectiveness and stealthiness, respectively. We observe that this loss function is also only related to the final layer F_{final}. With an SDN, the sample x has a high chance to exit the model earlier than the final layer, and will not be affected by the flipped bits in F_{final}.

Attack II: the adversary flips bits in arbitrary layers based on a more sophisticated search method, as exploited in ProFlip [8]. Specifically, the adversary selects a trigger Δ. Given a clean input sample x and the target class t, the adversary aims to search for critical bits in all the layers to inject the backdoor. This process is also modeled as an optimization problem with the following loss function:

$$\mathcal{L}_{ProFlip} = \mathcal{L}_{ce}(F_{final}(x+\Delta);t) - \mathcal{N}(x+\Delta), \quad (4)$$

where $\mathcal{N}$ denotes the salient neurons, determined by the target label through conducting the Jacobian Saliency Map Attack (JSMA) [43]. Particularly, the adversary calculates the gradients of the model inference $F_{final}(x)$ w.r.t. each neuron output, and then selects neurons with top gradient values as $\mathcal{N}$.

We observe that the joint impact of all the searched bits can only affect the output of F_{final}. Samples that exit earlier from the ICs might still give the correct prediction results.

Attack III: based on the above analysis, we conclude that the basic BFAs that flip bits in certain layers cannot break the basic SDN. So we further assume a stronger adversary, who knows our Aegis mechanism and aims to design an adaptive strategy to break it. He may design a loss function $\mathcal{L}^*$ that considers all the ICs to optimize. However, breaking our defense is still difficult, as explained below:

(1) For adaptive attacks based on Attack I (i.e., adaptive TBT and TA-LBF), considering all the ICs could identify the vulnerable bits that can affect each exit. However, this could significantly increase the attack cost (i.e., the number of bits to flip). In particular, to design an adaptive TBT, we consider the following loss function, which includes the ICs as well:

$$\mathcal{L}^*_{TBT} = \mathcal{L}_{TBT} + \sum_{i=1}^{N} \mathcal{L}_{ce}(C_i(x);l) + \mathcal{L}_{ce}(C_i(x+\Delta);t). \quad (5)$$

In this case, the adversary flips bits in the final layer of each IC as well as the final layer of M. Recall that in the TBT algorithm in Section 2.1, the adversary sets a fixed number of candidate parameters in the final layer as w_b. For each candidate parameter, the adversary figures out several bits to flip. This implies that the number of flipped bits is positively correlated with w_b. In adaptive scenarios, since each IC needs to be attacked, the adversary needs to modify w_b parameters in the final layer of each IC. This results in the scale of $n \times w_b$ parameters for compromising, which is a huge cost for the adversary, especially when n is large.

The adaptive TA-LBF can also be designed in a similar way by including the ICs in the loss function as follows:

$$\mathcal{L}^*_{TA-LBF} = \mathcal{L}_{TA-LBF} + \sum_{i=1}^{N} \mathcal{L}_1(x;l;t;C_i) + \mathcal{L}_2(x;l;t;C_i). \tag{6}$$

Increasing the number of candidate layers also results in a larger number of bits to flip, since the adversary needs to ensure all the exits are affected.

(2) For adaptive attacks based on Attack II (i.e., adaptive ProFlip), although the adversary considers all the ICs, he might still fail to attack each exit. Samples exiting from the unattacked ICs are not affected, and the attacks are mitigated. In particular, for the adaptive ProFlip attack, the adversary optimizes the following loss function:

$$\mathcal{L}^*_{ProFlip} = \mathcal{L}_{ce}(F_{final}(x);t) + \sum_{i=1}^{N} \mathcal{L}_{ce}(C_i(x);t) - \widetilde{\mathcal{N}}(x+\Delta). \tag{7}$$

Different from $\mathcal{N}$, the adversary considers all ICs as well as the final layer, i.e., $\sum_{i=1}^{N} C_i(x) + F_{final}(x)$. Particularly, the adversary adopts JSMA to calculate the gradients of $\sum_{i=1}^{N} C_i(x) + F_{final}(x)$ w.r.t each neuron outputs, then select neurons with top gradients, denoted as $\widetilde{\mathcal{N}}$.

The adversary selects the optimal parameter to modify through optimizing Eq. 7, and the optimal parameter might locate in any layer. Here, the determination of the optimal parameter is highly dependent on x given t, Δ, and the target model. For example, on CIFAR-100 and VGG16, we repeat the ProFlip attack 100 times, and randomly select 256 different input samples x for each time. For each independent attack, we observe the optimal parameter could be determined in different layers of $\hat{M}$. We assume the optimal parameter is located in the i-th layer. In this case, all ICs attached before the i-th layer are not destroyed by the adversary. Thus the samples exiting from these ICs are not affected. Besides, for the ICs attached after the i-th layer, our `ROB` could reduce the impacts brought by the flipped bits. Another case is that the optimal parameter is located in an IC. Only the IC is attacked by the adversary, and all other ICs are not affected.

5 Evaluation

We evaluate `Aegis` on different targeted BFAs: two backdoor attacks (TBT and TA-LBF) and one sample-wise attack (ProFlip). We also consider many adaptive attacks. All experiments are conducted on a machine with an Intel Xeon Gold 6154 CPU and 8 NVIDIA Tesla V100 GPUs.

5.1 Experimental Setup

Datasets and models. We conduct our experiments on four widely-used datasets with two DNN structures: VGG16 [56] and ResNet32 [16].

- CIFAR-10 [30]: This dataset contains 50,000 training images and 10,000 test images. Each image has a size of $32 \times 32 \times 3$ and belongs to one of 10 classes.
- CIFAR-100 [30]: This has the same number of training and testing images as CIFAR-10. Each image also has the same size as CIFAR-10, but belongs to one of 100 classes.
- STL-10 [9]: This dataset contains 50,00 training images and 8,000 test images. We note that it also contains 100,000 unlabeled images. Each image has a size of $96 \times 96 \times 3$ and belongs to one of 10 classes.
- Tiny-ImageNet [1]: This dataset is a simplified version of ImageNet consisting of color images with a size of $64 \times 64 \times 3$ belonging to 200 classes. Each class has 500 training images and 50 testing images.

Inspired by [3], we strictly separate the training and testing data without any overlap. In particular, (1) CIFAR-10: we follow [7, 19, 42, 60] to select 50,000/10,000 images for training/testing. (2) CIFAR-100: we follow [5, 28] to select 50,000/10,000 images for training/testing. (3) STL-10: we follow [58, 59] to select 5,000/8,000 images for training/testing. (4) Tiny-ImageNet: we follow [59, 60] to select 100,000/10,000 images for training/testing.

Hyperparameters. As mentioned in Section 4.2.2, τ and q affect the early-exit distribution and model accuracy. For generalization on unseen data and avoiding the selection of biased hyperparameters, we tune hyperparameters on training data guided by two goals: (1) making early-exits uniformly distributed to prevent the attacker from targeting only those popular exits; (2) maintaining high ACC on benign samples. Table 1 list the values of these hyperparameters. Note that we tune these hyperparameters without considering any specific attacks. They are general and fixed to mitigate all attacks in our consideration. We also evaluate the sensitivity of these hyperparameters and find the mitigation results are stable to hyperparameters that meet the two goals (see Appendix A).

Table 1: Values of τ and q on our datasets and models.

Dataset	Model	τ	q
CIFAR-10	ResNet32	0.95	3
	VGG16	0.95	3
CIFAR-100	ResNet32	0.90	5
	VGG16	0.80	3
STL-10	ResNet32	0.95	5
	VGG16	0.95	3
Tiny-ImageNet	ResNet32	0.90	3
	VGG16	0.95	4

We also conduct `ROB` for the ICs. The only randomness of Aegis is from the random selection of ICs in inference. We repeated each experiment 10 times and the ASR variance is below 2% which does not affect our conclusion.

Baselines. We compare `Aegis` with the state-of-the-art defense methods BIN [19] and RA-BNN [49]. We also compare `Aegis` with the basic SDN [27] to demonstrate the effectiveness of `DESDN` and `ROB` mechanisms. For a fair comparison, we slightly modify SDN to make sure `Aegis` uses the same structure and hyperparameters as SDN, except for the `DESDN` and `ROB` mechanisms. Besides, we compare `Aegis` with the baseline models (BASE) with no defense.

5.2 Model Utility Evaluation

A qualified defense method should preserve the utility, i.e. tiny model accuracy (ACC) drop. Table 2 compares the impacts of different methods on ACC. We observe that `Aegis` slightly degrades ACC by approximately less than 2%, while BIN and RA-BNN have much more ACC degradation, i.e., roughly $2 \sim 12\%$ and $2 \sim 7\%$, respectively. This is due to BIN and RA-BNN adopting an aggressive binarization on weights (each parameter occupies only 1 bit) or activation function outputs which harm the model ACC. We also notice that SDN has comparable ACC with `Aegis`, which validates that the `DESDN` mechanisms in `Aegis` do not affect the model utility. In summary, `Aegis` could preserve the model utility.

Table 2: Model ACC influence evaluation.

Dataset	Model	BASE ACC (%)	Δ ACC (%)			
			BIN	RA-BNN	SDN	`Aegis`
CIFAR-10	ResNet32	92.79	-2.26	-1.71	-1.27	**-1.26**
	VGG16	93.61	-1.26	-1.19	-1.72	**-0.67**
CIFAR-100	ResNet32	66.13	-4.38	-2.47	-2.54	**-1.96**
	VGG16	72.85	-4.14	-2.08	-1.97	**-1.90**
STL-10	ResNet32	74.80	-4.09	-3.85	-2.80	**-0.90**
	VGG16	79.51	-1.41	-1.39	-1.35	**-1.02**
Tiny-ImageNet	ResNet32	54.58	-11.16	-6.31	-3.87	**-1.92**
	VGG16	60.51	-4.18	-4.07	-0.39	**-0.28**

5.3 Mitigating Targeted Attacks

We evaluate the defense effectiveness of `Aegis` against the state-of-the-art targeted attacks including two backdoors targeted BFAs (TBT [46] and ProFlip [8]) and one sample-wise targeted BFA (TA-LBF [4]). We reproduce these BFAs with their open-sourced code and the recommended parameters (Appendix B) and list the visual results (e.g. triggers and samples) in Appendix C. The possible adaptive attacks based on these BFAs are evaluated in Section 5.4.

Comparison metrics. While applying an attack to all defense methods, we compare the attack success rate (ASR) of each defense method. For fair comparisons, we make sure the given attack pays the same attack cost on each defense method: flipping the same number of bits. We denote the number of flipped bits as N_b and consider different N_b with two steps.

First, we restrict the bit flipping number limit (N_b) to 50 for all attacks. Such a bit flipping number limit is set as 24 in the state-of-the-art Rowhammer attack on cloud platforms [24]: it tests a batch of high-quality dual inline memory modules (DIMMs) and reveals that flipping 24 bits needs a significantly long time (several hours). We also confirm the feasibility of performing BFAs on physical systems (in Section 5.7) to illustrate that flipping 50 bits is very difficult for attackers. Furthermore, we relax the restriction on N_b (i.e. 500) and evaluate the ASR for comprehensiveness.

In the following, we first evaluate all defense methods under TBT and TA-LBF, as they all flip bits in the final layer of the target model. We then evaluate all defense methods under ProFlip, which could flip bits in any layer of the target model.

Mitigating TBT. Table 3 shows the defense results against TBT. `Aegis` can significantly decrease the ASR and outperform other methods. In most cases, `Aegis` can decrease the ASR to less than 20%, which is significantly lower than others. In contrast, we note BIN and RA-BNN even perform worse than BASE in some cases. For example, on CIFAR-10 and VGG16, the ASR for BASE is 71.1%, which is lower than that of BIN (90.4%) and RA-BNN (82.9%). This means the defenses designed for untargeted BFAs might make models even more vulnerable to targeted BFAs.

Table 3: Evaluation results of ASR against TBT.

Dataset	Model	ASR (%)				
		BASE	BIN	RA-BNN	SDN	`Aegis`
CIFAR-10	ResNet32	70.7	94.8	74.5	**16.3**	19.9
	VGG16	71.1	90.4	82.9	42.6	**36.0**
CIFAR-100	ResNet32	95.8	99.8	25.5	20.5	**10.8**
	VGG16	65.9	58.4	47.4	53.8	**10.6**
STL-10	ResNet32	100.0	72.5	29.4	47.1	**13.0**
	VGG16	64.1	99.7	88.0	**9.0**	10.5
Tiny-ImageNet	ResNet32	100.0	63.3	31.4	65.8	**27.9**
	VGG16	69.7	72.3	40.2	48.9	**10.1**

Mitigating TA-LBF. Table 4 shows the ASR on all datasets and models against TA-LBF. Overall, `Aegis` could effectively mitigate TA-LBF and outperform other methods. In most cases, `Aegis` can limit ASR below 10.0%, which is much smaller than others. For example, on STL-10 and ResNet32, `Aegis` limits the ASR to 9.6% while the ASR for BASE, BIN, RA-BNN, and SDN is 100.0%, 100.0%, 100.0% and 47.7%, respectively. We find the highest ASR for `Aegis` is 20.1% on Tiny-ImageNet and ResNet32 which is still much smaller than the ASR for other cases.

Table 4: Evaluation results of ASR against TA-LBF.

Dataset	Model	ASR (%)				
		BASE	BIN	RA-BNN	SDN	`Aegis`
CIFAR-10	ResNet32	100.0	100.0	100.0	**3.5**	6.3
	VGG16	57.6	100.0	100.0	1.1	**0.3**
CIFAR-100	ResNet32	100.0	100.0	100.0	38.0	**16.4**
	VGG16	56.4	100.0	100.0	19.4	**4.4**
STL-10	ResNet32	100.0	100.0	100.0	47.7	**9.6**
	VGG16	81.4	99.7	98.7	**0.3**	2.0
Tiny-ImageNet	ResNet32	100.0	100.0	100.0	71.1	**20.1**
	VGG16	51.8	98.1	90.7	27.2	**17.3**

Summary for the evaluation of TBT and TA-LBF. We summarize the defense evaluation against TBT and TA-LBF as follows. These two attacks only flip bits in the final layer, which could not affect ICs. `Aegis` can keep the model ACC well and let many input samples early-exit from ICs to effectively mitigate TBT and TA-LBF. We notice that SDN also performs better than BASE, BIN, and RA-BNN in most cases. However, SDN uses a static multi-exit mechanism, making it less effective than `Aegis` for adaptive attacks (Section 5.4).

Table 5: Evaluation results of ASR against ProFlip.

Dataset	Model	ASR (%)				
		BASE	BIN	RA-BNN	SDN	**Aegis**
CIFAR-10	ResNet32	96.9	99.4	90.6	47.3	**19.8**
	VGG16	88.2	78.6	84.6	70.5	**28.9**
CIFAR-100	ResNet32	89.8	100.0	82.9	58.3	**19.2**
	VGG16	80.0	80.4	76.5	64.9	**20.3**
STL-10	ResNet32	77.4	52.4	91.2	58.1	**33.9**
	VGG16	87.2	96.0	90.3	19.9	**18.7**
Tiny-ImageNet	ResNet32	99.1	82.5	80.4	75.0	**20.1**
	VGG16	88.2	44.1	39.2	26.8	**15.6**

Mitigating ProFlip. Different from TBT and TA-LBF, ProFlip does not restrict the layer but uses an optimization method to flip critical bits in arbitrary layers. Tables 5 shows the results in which `Aegis` can limit the ASR below 30% in most cases. On the contrary, the ASR for BASE, BIN, and RA-BNN is higher than 70% in most cases. For example, on CIFAR-10 dataset with VGG16, `Aegis` limits the ASR to 28.9%. However, the adversary acquires 88.2%, 78.6%, and 84.6% ASR for BASE, BIN, and RA-BNN. `Aegis` makes that the flipped bits could not effectively affect samples that exit earlier. This is also why SDN has comparable defense performance in some cases.

Evaluation with more bits flipped. Here we do not restrict N_b to 50 but consider larger N_b to compare `Aegis` with other methods. We take CIFAR-100 and VGG16 as an example to evaluate when N_b increases to up to 500 for all defense methods. Evaluation results are shown in Figure 4. Compared with existing defense methods, we observe the ASR for `Aegis` is still limited at a low level for all attacks. For TBT (Figure 4 (a)) and TA-LBF (Figure 4 (b)), even the attacker could flip as many as 500 bits, the ASR is still less than 13%, which proves the defense effectiveness of our method. For ProFlip (Figure 4 (c)), ASR increases slowly with more bits flipped and `Aegis` can clearly outperform all baselines. Even if the attacker could flip as many as 500 bits, `Aegis` can restrict the ASR of ProFlip to 58.3% while ASR for other cases is already 100% with significantly fewer bits flipped.

Note that when more bits are flipped, `Aegis` clearly outperforms SDN against all three attacks. Especially, considering ProFlip, the ASR achieves almost 100% when about 200$\sim$300 bits are flipped. Thus, we claim that simply deploying a multi-exit DNN structure on a target vanilla model cannot sufficiently defeat these targeted BFAs even without considering their adaptive attacks. Also, `Aegis` includes two components, and the ablation study to evaluate their defense effectiveness respectively is given in Section 5.6.

5.4 Mitigating Adaptive Attacks

Beyond basic adversaries, any effective defense should also be capable of withstanding any adaptive attackers who are aware of the existence and mechanism of the defense. We consider a sophisticated adversary who knows the detail of our defense mechanism and aims to design an adaptive strategy to break it. We consider crafting such advanced attacks from the three state-of-the-art attack methods (i.e., TBT, TA-LBF, and ProFlip). As analyzed in Section 4.4, the adversary knows all details of `Aegis` and tries to attack all ICs to increase the ASR regardless of where the input samples exit. He can design new loss functions dedicated to `Aegis` by adding optimization terms for all ICs. Thus, the adaptive TBT and TA-LBF attacks will flip bits in the final layer of all ICs in addition to the final layer of the model. All other attack settings and models are the same as in the previous sections.

In the following, we evaluate the effectiveness of `Aegis` against these adaptive attack scenarios. We still first set N_b to 50 for all attacks. Then, we relax the restriction of N_b and evaluate the ASR when the adversary could flip more bits.

Baselines. Since BIN is proven to make the victim model even more vulnerable to targeted BFAs, we do not consider it as a baseline defense and mainly compare with SDN. Both `Aegis` and SDN have ICs attached to the hidden layers for an early-exit mechanism, enabling the adversary to design a similar dedicated adaptive attack against them. We also report the ASR of BASE to reflect the defense effectiveness.

Adaptive TBT. Table 6 shows the ASR of each defense method. We observe that SDN can be defeated by the adaptive TBT attack. By including all ICs of the SDN model to perform the adaptive TBT attack, the ASR of SDN is even higher than that of BASE in many cases. In contrast, our `Aegis` can restrict the ASR below 40% in most cases. This is because the adaptive TBT attack will flip bits for critical parameters in ICs and the model's final layers to manipulate inference on all exits of the SDN model. However, `Aegis` still effectively mitigates this strategy due to the random exit mechanism.

Table 6: Evaluation results of ASR against adaptive TBT.

Dataset	Model	ASR (%)		
		BASE	SDN	**Aegis**
CIFAR-10	ResNet32	70.7	37.2	**31.1**
	VGG16	71.1	86.5	**58.1**
CIFAR-100	ResNet32	95.8	79.3	**49.7**
	VGG16	65.9	85.9	**44.8**
STL-10	ResNet32	100.0	35.0	**31.8**
	VGG16	64.1	93.0	**27.0**
Tiny-ImageNet	ResNet32	100.0	96.3	**28.2**
	VGG16	69.7	63.4	**54.4**

Adaptive TA-LBF. Table 7 reports the defense results against adaptive TA-LBF, indicating the effectiveness of `Aegis`. Similarly, SDN becomes even more vulnerable than BASE under the adaptive TA-LBF attack. We analyze that compared with the basic SDN, `Aegis` still effectively mitigates the adaptive TA-LBF attack due to the `DESDN`.

Summary of evaluating adaptive TBT and TA-LBF. We analyze why `Aegis` is better than SDN against the adaptive TBT and TA-LBF is the `DESDN` scheme. Since SDN uses a static multi-exit mechanism, input samples have a stable exit pattern, i.e. most samples exit from a few fixed ICs. We give an example on CIFAR-100 and VGG16 to list the proportion

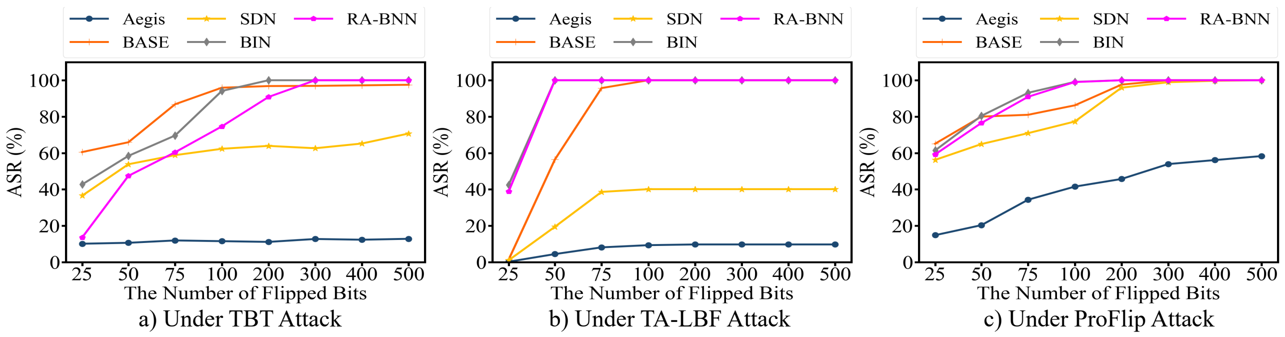


Figure 4: Comparison between `Aegis` and other defense methods on CIFAR-100 with VGG16 when more bits are flipped. `Aegis` performs better than other methods: the ASR for `Aegis` is much lower than others under different numbers of flipped bits.

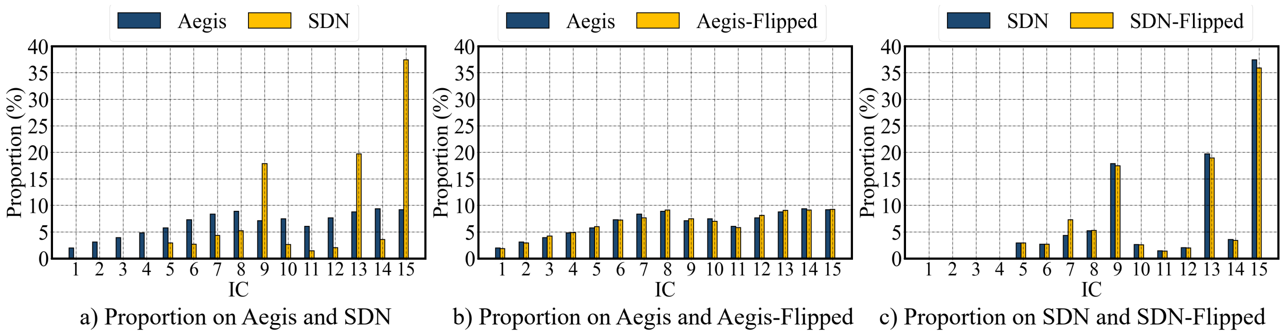


Figure 5: The proportion of samples exit from different ICs or final layer (15 denotes the final layer) on CIFAR-100 and VGG16. Samples exit more uniformly in `Aegis` than SDN, even under BFA attacks (`Aegis`-Flipped and SDN-Flipped).

Table 7: Evaluation results of ASR against adaptive TA-LBF.

Dataset	Model	ASR (%)		
		BASE	**SDN**	**`Aegis`**
CIFAR-10	ResNet32	100.0	99.1	**60.8**
	VGG16	70.2	89.3	**50.3**
CIFAR-100	ResNet32	100.0	100.0	**26.4**
	VGG16	56.4	78.2	**44.8**
STL-10	ResNet32	100.0	100.0	**10.2**
	VGG16	81.4	89.9	**26.8**
Tiny-ImageNet	ResNet32	100.0	100.0	**16.2**
	VGG16	51.8	90.4	**15.0**

of samples exiting pattern in Figure 5. In Figure 5 (a), we observe that SDN lets more than 75% samples exit from three exits, i.e. IC_9, IC_{13}, and IC_{15}. Once the adversary modifies TBT and TA-LBF adaptively by including the loss for all ICs, its optimization process will utilize this exit pattern and locate the adaptive critical bits in these ICs to perform BFAs. `Aegis` adopts the `DESDN` mechanism to make input samples exit from all ICs uniformly. Even if the adversary adaptively attacks all ICs, the uniformly distributed exits will increase the attack cost (more bits to flip), thus enhancing security. Therefore, under the same N_b, `Aegis` significantly outperforms SDN.

We further consider whether the exit distributions of these defenses can be affected by the adaptive attack. Taking TA-LBF as an example, we denote the compromised SDN and `Aegis` as SDN-Flipped and `Aegis`-Flipped, respectively. Figures 5 (b) and (c) show that the original and flipped models of each method have similar exit distributions. Such results reveal that the distribution of `Aegis` is still uniform under the attack, giving the adversary no chances to utilize the exit pattern to locate the critical parameters of critical ICs. In contrast, the flipped SDN model exhibits the same exit distribution (i.e., vulnerability) as the original one.

Adaptive ProFlip. Table 8 shows the defense results for the adaptive ProFlip attack. Compared with SDN and BASE, the ASR of `Aegis` is always significantly lower on all datasets and models. Taking CIFAR-100 and ResNet32 as an example, the ASR of `Aegis` is 25.8%, while the ASR of SDN and BASE is 69.1% and 89.8% respectively. We analyze the main reason: with the optimization function for determining the critical parameters, the flipped bits are more likely to be concentrated in one layer. Thus, the adversary cannot effectively affect ICs before the modified layer, making `Aegis` resilient against this attack. The reason why SDN has relatively good defense results in some cases also comes from this. However, SDN is much inferior to `Aegis` as it only uses a static early exit mechanism. Such a static mechanism makes the number of affected ICs in the SDN greater than that in `Aegis`. Besides `DESDN`, our `ROB` contributes to the defense, which will be evaluated in Section 5.6.

Table 8: Evaluation results of ASR against adaptive ProFlip.

Dataset	Model	ASR (%)		
		BASE	**SDN**	**`Aegis`**
CIFAR-10	ResNet32	96.9	74.2	**38.4**
	VGG16	88.2	79.1	**43.6**
CIFAR-100	ResNet32	89.8	69.1	**25.8**
	VGG16	80.0	92.4	**33.7**
STL-10	ResNet32	77.4	57.8	**41.3**
	VGG16	87.2	87.5	**34.5**
Tiny-ImageNet	ResNet32	99.1	64.4	**36.1**
	VGG16	88.2	73.1	**40.8**

Targeting shallow hidden layers. We further consider another adaptive attack, which focuses on shallow hidden layers to flip bits (denoted as **Shallow**). Note that only ProFlip could be extended to achieve such an adaptive attack since TBT and TA-LBF must modify parameters connected to the target class, which are located in the last dense layer.

In particular, we modify ProFlip to choose critical bits among the first three hidden layers. Table 9 shows the results of Shallow. We observe that attacking shallow layers is not effective in bypassing `Aegis`. Indeed, flipping bits in shallow layers cannot guarantee successful targeted attacks on the following hidden layers' outputs while `Aegis` lets samples randomly exit from arbitrary layers to limit ASR. Besides, targeting shallow layers may significantly decrease clean ACC ($6.2-27.8\%$), while the original ProFlip only degrades 2%. This makes such adaptive attacks easy to be detected.

Table 9: Evaluation results of attacking shallow layers.

Dataset	Model	ASR (%)			Δ ACC (%)
		BASE	`Aegis`	Shallow	
CIFAR-10	ResNet32	96.9	38.4	40.5	**-18.7**
	VGG16	88.2	43.6	49.5	**-21.5**
CIFAR-100	ResNet32	89.8	25.8	22.8	**-6.2**
	VGG16	80.0	33.7	37.0	**-8.8**
STL-10	ResNet32	77.4	41.3	48.4	**-23.1**
	VGG16	87.2	34.5	51.7	**-8.9**
Tiny-ImageNet	ResNet32	99.1	36.1	30.3	**-27.8**
	VGG16	88.2	40.8	51.3	**-10.4**

Evaluation with more bits flipped. We further evaluate the defense methods with up to 500 bits flipped for comprehensiveness. We conduct experiments on CIFAR-100 and VGG16 and adopt the adaptive TBT, TA-LBF, and ProFlip attacks. Results are shown in Figure 6. Compared with existing defense methods, under the same value of N_b, we observe the ASR of `Aegis` is always the lowest against all attacks.

5.5 Evaluation of Model Size

`Aegis` can increase the model size during deployment. We evaluate the model size increase for all datasets and model structures used by us and find models ranging from 6.1MB to 97.6MB. We find that the size increase depends on the datasets and model structures. Particularly, `Aegis` introduces a tiny increase for VGG16. For example, on CIFAR-10, the size of VGG16 is 58.3MB and 65.6MB for BASE and `Aegis`, respectively. In contrast, ResNet is a small model so the increase is relatively larger. For instance, also on CIFAR-10, the size of VGG16 is 1.9MB and 6.1MB for BASE and `Aegis`. We emphasize that such a size increase is not a bottleneck for model deployment in practice. We find that for common embedded devices (e.g, Nvidia Jetson Nano), the memory capacity is usually at the GB level (far more sufficient to support `Aegis`). Besides, `Aegis` significantly improves the inference efficiency with almost no accuracy drop since most samples could early-exit from the network. Such benefits at the cost of acceptable size increase are very attractive for inference applications at the edge.

To validate the above points, we deploy our `Aegis` on two real-world widely-used edge devices: (1) Nvidia Jetson Nano with 4GB memory and 16GB storage; (2) Raspberry Pi 4 with 4GB memory and 32GB storage. The size increase is totally affordable for these two devices. We further evaluate the inference acceleration brought by `Aegis`. We observe the average inference time is $46.1-59.4\%$ of the original model, which is a big improvement.

5.6 Ablation Study on `ROB`

We verify the effectiveness of `ROB`. We compare `Aegis` with the setting without `ROB`, i.e., just `DESDN`. Note that `ROB` aims to help ICs adapt to the adversarial scenario where bits in the layers attached by ICs are flipped. Therefore, we choose ProFlip to evaluate the effectiveness of `ROB`, as it can flip bits in the layers attached by ICs. Table 10 shows that `ROB` effectively improves defense results. For the basic attacks, the ASR of `Aegis` is $3\%-13\%$ less than that of `DESDN`. For the adaptive attacks, `Aegis` also performs better than `DESDN` by reducing $4\%-15\%$ of ASR. Overall, we prove that `ROB` effectively contributes to mitigating targeted BFAs.

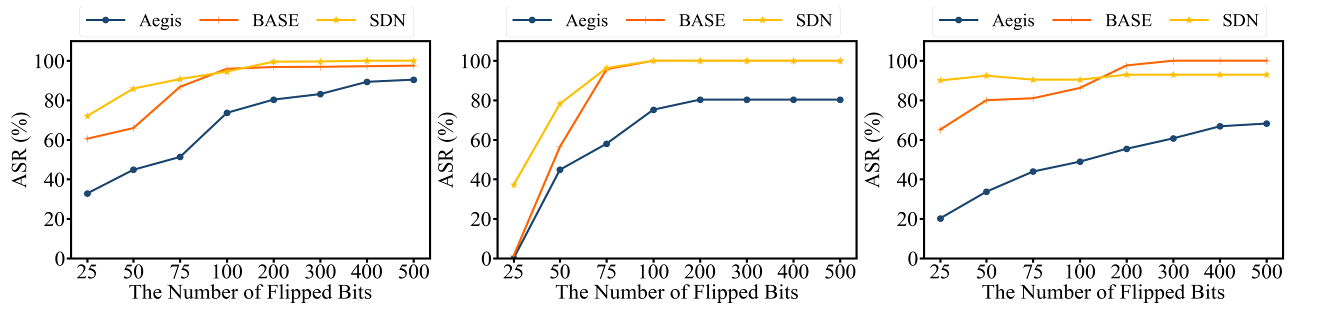
Table 10: Impact of `ROB` on basic and adaptive ProFlip.

Dataset	Model	ASR (%)			
		Basic ProFlip		Adaptive ProFlip	
		`DESDN`	`Aegis`	`DESDN`	`Aegis`
CIFAR-10	ResNet32	24.1	**19.8**	45.1	**38.4**
	VGG16	33.7	**28.9**	49.4	**43.6**
CIFAR-100	ResNet32	29.8	**19.2**	39.2	**25.8**
	VGG16	28.7	**20.3**	51.4	**33.7**
STL-10	ResNet32	36.2	**33.9**	45.0	**41.3**
	VGG16	22.9	**18.7**	39.6	**34.5**
Tiny-ImageNet	ResNet32	33.4	**20.1**	45.4	**36.1**
	VGG16	22.9	**15.6**	50.2	**40.8**

5.7 Attack Feasibility Analysis

We validate the feasibility of BFAs (more specifically, TBT attack), and use ResNet32 and Tiny-ImageNet as an example. The main idea is to adopt the DeepSteal [44] technique, which provides a memory massaging mechanism to realize Rowhammer [29] attack, and is able to flip multiple bits within a 4KB page. This satisfies TBT's requirement, which targets one row of weights in the model's final layer and needs flipping multiple bits within a 4KB page. This mechanism massages a page multiple times using memory swapping (the feature of swapping physical pages from DRAM to the disk swap space under memory pressure and then swapping them back when needed by the processor). Below we describe the details.

Step 1: evicting victim pages. This step aims to evict the victim's pages from the main memory to swap space such that they can be relocated by the OS when they are accessed by the victim next time. To accomplish this, adversaries first allocate a large chunk of memory using `mmap` with the `MAP_POPULATE`



a) Under Adaptive TBT Attack b) Under Adaptive TA-LBF Attack c) Under Adaptive ProFlip Attack

Figure 6: On CIFAR-100 and VGG16, we compare `Aegis` with other defense methods under different values of N_b. Even the adversary significantly increases N_b, `Aegis` still outperforms others in the adaptive scenarios.

flag. This triggers the OS to evict other data (including the victim's pages) from the main memory to the swap space. Thus, we can occupy most of the physical memory space with victim pages stored in swap space.

Step 2: releasing pages. This step aims to systematically release the occupied pages to enforce the desired relocation of the victim's pages. In detail, adversaries create a list of potential pages for the victim to occupy as aggressors during the attack. At each round, adversaries choose a predetermined number of pages from the list and release the selected pages by calling `munmap`.

Step 3: deterministic relocation. This step aims to place victim pages in predetermined locations to create an appropriate memory layout for Rowhammer. It also ensures that the victim page location is known to adversaries so that they correlate flipped bits with exact data in the victim domain. Adversaries follow DeepSteal [44] to exploit the per-core page-frame cache structure to manipulate the operating system page allocation, which allows them to control where the victim pages are relocated.

After the above steps, adversaries mount Rowhammer [29] to flip bits in the victim pages placed in the appropriate locations. Since TBT targets one row of weights in a model's final layer which requires flipping multiple bits within a 4KB page, adversaries iterate the aforementioned steps until all target bits are flipped. Note that after each iteration, the weight page (with bit flips) will be swapped to the disk under memory pressure. When this page is needed again, it is swapped back. As a consequence, bit flips will occur with this operation. When the weight page is swapped back, it has a high probability to be put into a new location in the memory where a different bit can be flipped. Then adversaries perform Rowhammer again to this page. Adversaries iterate the entire process until all the required bits are flipped.

Evaluation results. Using the above technique, adversaries are able to flip 10 bits in the target model to achieve TBT. The index of the flipped bits can be found in Appendix E. For base models, flipping these bits can achieve an ASR of 77.8%. Now with our `Aegis`, the TBT attack can only get an ASR of 2.0%. This confirms the effectiveness of `Aegis`.

We further measure the attack cost. Flipping one bit takes about tens or even hundreds of seconds. Therefore, flipping more bits requires a much higher attack cost. DeepHammer [64] assumes that the maximum number of bits the adversary is allowed to flip is 24. To highlight the effectiveness of our defense, we consider a more powerful adversary who is allowed to flip $N_b = 50$ bits (taking several hours) in Section 5.3. Our `Aegis` is still effective against such an attack. Furthermore, we assume an unrealistically strong attack ($N_b = 500$), and `Aegis` is still able to defeat it. Figures 4 and 6 show the evaluation results for non-adaptive and adaptive attacks under this attack cost.

6 Discussion and Future Work

Difference between `ROB` and adversarial training. Both adversarial training [41] and our `ROB` aim to improve the model robustness by modifying model parameters but they are significantly different. Adversarial training aims to specifically defeat adversarial attacks by using perturbed samples generated from a clean model. Various previous work [19, 45] have proved that directly using adversarial training cannot mitigate existing BFAs. In contrast, `ROB` improves robustness by considering a compromised model. By considering the effects brought by flipping critical bits of a target model, `ROB` can particularly increase the resistance against BFAs.

Floating-point DNN models. To the best of our knowledge, all state-of-the-art BFAs [4, 8, 45, 46] only focus on attacking quantized models. Therefore, we follow these existing works to evaluate our defense methods on quantized models in this paper. In fact, since `Aegis` is experimented with in a non-intrusive fashion, protecting the floating-point DNN model is also feasible. We leave this experimentation as our future work.

Potential defense effects against untargeted BFAs. As indicated in Section 3, mitigating untargeted BFAs is beyond the scope of this paper. However, we still evaluate `Aegis` against a state-of-the-art untargeted BFA [45] for comprehensiveness. As shown in Appendix D, results show that `Aegis` could also mitigate untargeted BFA effectively by significantly increasing the attack cost (i.e. the number of flipped bits). We consider this as future work.

7 Conclusion

We propose Aegis, a novel mitigation methodology against targeted bit-flip attacks. With a novel design of DESDN, we randomly select ICs for inference, enabling input samples to early-exit from them and effectively obfuscate the adversary. We further propose ROB to improve IC robustness. We conduct extensive experiments with four mainstream datasets and two DNN structures to show that Aegis can mitigate various state-of-the-art targeted attacks as well as their adaptive versions, and significantly outperform existing defense methods.

Acknowledgments

This work is supported by the National Key R&D Program of China (2022YFB3105202), National Natural Science Foundation of China (62106127, 62132011, 61972224), NSFOCUS (2022671026), Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121- 0006, and Ant Group through CCF-Ant Innovative Research Program No. RF2021002.

References

[1] Tiny-Imagenet. tiny-imagenet.herokuapp.com.

[2] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security*, 2018.

[3] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *USENIX Security*, 2022.

[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 *ICLR*, 2021.

[5] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. In *NeurIPS*, 2021.

[6] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *ACM CCS*, 2018.

[7] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE SP*, 2017.

[8] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *ICCV*, 2021.

[9] Adam Coates, Andrew Y. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[10] 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.

[11] Tian Dong, Ziyuan Zhang, Han Qiu, Tianwei Zhang, Hewu Li, and Terry Wang. Mind your heart: Stealthy backdoor attack on dynamic deep neural network in edge computing. *arXiv preprint arXiv:2212.11751*, 2022.

[12] 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.

[13] Max Ghiglione and Vittorio Serra. Opportunities and challenges of ai on satellite processing units. In *Proceedings of the 19th ACM International Conference on Computing Frontiers*, pages 221–224, 2022.

[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[15] Yanan Guo, Liang Liu, Yueqiang Cheng, Youtao Zhang, and Jun Yang. Modelshield: A generic and portable framework extension for defending bit-flip based adversarial weight attacks. In *ICCD*, 2021.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[17] Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *ACSAC*, 2019.

[18] Zecheng He, Tianwei Zhang, and Ruby B Lee. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 2020.

[19] 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.

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

[21] Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. In *ICLR*, 2020.

[22] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *ICLR*, 2018.

[23] Shehzeen Hussain, Paarth Neekhara, Shlomo Dubnov, Julian J. McAuley, and Farinaz Koushanfar. Waveguard: Understanding and mitigating audio adversarial examples. In *USENIX Security*, 2021.

[24] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *IEEE SP*, 2022.

[25] Mojan Javaheripi and Farinaz Koushanfar. HASHTAG: hash signatures for online detection of fault-injection attacks on deep neural networks. In *IEEE/ACM ICCAD*, 2021.

[26] Yichen Jiang, Huifeng Zhu, Haoqi Shan, Xiaolong Guo, Xuan Zhang, and Yier Jin. Trrscope: Understanding target row refresh mechanism for modern ddr protection. In *IEEE HOST*, 2021.

[27] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, 2019.

[28] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *NeurIPS*, 2020.

[29] 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 ISCA*, 2014.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[31] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. RADAR: run-time adversarial weight attack detection and accuracy recovery. In *DATE*, 2021.

[32] 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.

[33] Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. Backdoor attacks on pre-trained models by layerwise weight poisoning. In *EMNLP*, 2021.

[34] 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.

[35] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745*, 2020.

[36] Yu Li, Min Li, Bo Luo, Ye Tian, and Qiang Xu. Deepdyve: Dynamic verification for deep neural networks. In *ACM CCS*, 2020.

[37] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *ICCV*, 2021.

[38] Qi Liu, Wujie Wen, and Yanzhi Wang. Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators. In *ICCAD*, 2020.

[39] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *ECCV*, 2020.

[40] Giulio Lovisotto, Henry Turner, Ivo Sluganovic, Martin Strohmeier, and Ivan Martinovic. SLAP: improving physical adversarial examples with short-lived adversarial perturbations. In *USENIX Security*, 2021.

[41] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[42] Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In *ICML*, 2019.

[43] 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.

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

[45] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *ICCV*, 2019.

[46] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. TBT: targeted neural network attack with bit trojan. In *CVPR*, 2020.

[47] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *TPAMI*, 2021.

[48] 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.

[49] 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. *arXiv preprint arXiv:2103.13813*, 2021.

[50] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 2015.

[52] Takami Sato, Junjie Shen, Ningfei Wang, Yunhan Jia, Xue Lin, and Qi Alfred Chen. Dirty road can attack: Security of deep learning based automated lane centering under physical-world attack. In *USENIX Security*, 2021.

[53] Giorgio Severi, Jim Meyer, Scott E. Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *USENIX Security*, 2021.

[54] Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2021.

[55] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE JBHI*, 2017.

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[57] Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. Hybrid batch attacks: Finding black-box adversarial examples with limited queries. In *USENIX Security*, 2020.

[58] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. In *NeurIPS*, 2013.

[59] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? In *NeurIPS*, 2020.

[60] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. BET: black-box efficient testing for convolutional neural networks. In *ISSTA*, 2022.

[61] 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.

[62] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

[63] Baoyuan Wu and Bernard Ghanem. ℓ_p-Box ADMM: A versatile framework for integer programming. *TPAMI*, 2019.

[64] 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*, 2020.

[65] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent backdoor attacks on deep neural networks. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2019.

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

[67] 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.

[68] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *CVPR*, 2020.

[69] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *NeurIPS*, 2020.

A Analysis of Hyperparameters

Tuning τ and q are not related to any specific attacks. We tune these two hyperparameters for two goals on clean models only, i.e. (1) making early-exits uniformly distributed to prevent the attacker from targeting only those popular exits; (2) maintaining high ACC on benign samples. Our results are stable to hyperparameters that meet these two requirements. We evaluate the sensitivity of these two hyperparameters on CIFAR-10 and ResNet32.

In particular, we first fix $q = 3$ and set τ to 0.93, 0.95, 0.97, respectively, to observe the sensitivity of τ on mitigating the attacks. Then, we fix $\tau = 0.95$, and set q to 2, 3, 4, respectively, to observe the sensitivity of q. Note that, these chosen values of τ and q could make early-exits uniformly distributed, as well as restrict the degradation of accuracy below 1.35%. Table 11 and 12 present the mitigation results considering different τ and q. For all three attacks, under both non-adaptive and adaptive settings, the ASR variance is below 2%. Such results reveal that the effectiveness of `Aegis` is stable if the chosen values of τ and q meet the two goals. In contrast, if we violate the two goals to tune the hyperparameters, the mitigation results should be hugely affected. For example, if we randomly set $\tau = 0.7$ and $q = 8$, most samples would early exit from the shallow layers. In this case, the ACC significantly degrades and the adaptive adversary could focus on the popular exits to improve the ASR.

Table 11: Evaluation results of the sensitivity on τ.

Attacks	τ	**ASR (%)**	
		Non-adaptive	**Adaptive**
TBT	0.93	19.1	30.2
	0.95	19.9	31.1
	0.97	21.1	31.6
TA-LBF	0.93	5.1	59.6
	0.95	6.3	60.8
	0.97	7.0	60.2
ProFlip	0.93	18.4	38.5
	0.95	19.8	38.4
	0.97	20.3	38.0

Table 12: Evaluation results of the sensitivity on q.

Attacks	q	**ASR (%)**	
		Non-adaptive	**Adaptive**
TBT	2	19.5	29.2
	3	19.9	31.1
	4	20.0	31.3
TA-LBF	2	6.5	59.8
	3	6.3	60.8
	4	4.7	61.2
ProFlip	2	19.4	38.3
	3	19.8	38.4
	4	19.0	38.6

B Hyperparameters of Attacks

TBT Hyperparameters. We follow the same settings in [46]. In particular, there are three hyperparameters: t, w_b, and TAP. t is the attack target class. We randomly select t in each independent experiment and employ SGD with a learning rate of 0.5 for 200 epochs. w_b is the number of weights to be flipped, and we set $w_b = 10$. TAP is the percentage of the area of the trigger trained by the attacker to the area of the input image, and we set the TAP to 9.76%.

TA-LBF Hyperparameters. We follow the same settings in [4]. There are three hyperparameters: N, k, and λ. N controls the effect on model accuracy and here we choose a fixed number 128 for N. k limits the number of flipped bits. Here we set the initial k as 5 and the maximum searching iteration for k as 6 for each attack. λ is a trade-off hyperparameter that keeps a balance between the attack effectiveness and the model utility. Here we set the initial λ as 100 and the maximum searching times as 8 for each attack. Note that for each dataset, we choose 1000 samples (each with a pre-assigned targeted label different from the ground-truth label) for TA-LBF attack. For each of the 10 classes in CIFAR-10 and STL-10, we randomly select 100 validation images from the other 9 classes. For CIFAR-100 and Tiny-ImageNet, we randomly choose 50 target classes, and for each of them randomly select 20 validation images from the other classes.

ProFlip Hyperparameters. We follow the same settings in [8]. In particular, there are two hyperparameters: t, k, and TAP. t controls the attack target class. We randomly select t in each independent experiment. In order to reduce the computational complexity, in each iteration, ProFlip determines the critical bits by traversing k points between the maximum and minimum values of the parameters. We choose $k = 20$ for CIFAR-10 and CIFAR-100, and $k = 10$ for STL-10 and Tiny-ImageNet. TAP is the percentage of the area of the trigger trained by the attacker to the area of the input image, and we set the TAP to 9.76%.

C Visualization of Triggers

We give several examples to show the triggers generated by TBT and ProFlip. Both traditional backdoor attacks [54, 65] and these two targeted BFAs generate triggers but they are significantly different. Traditional backdoor attacks tend to specify a pre-determined trigger. In contrast, TBT and ProFlip train a trigger from scratch to activate some specific neurons in DNNs. Thus, we can observe the triggers in Figure 7, 8 have no specific style.

D Mitigating Untargeted Attacks

Although `Aegis` is not designed to mitigate untargeted BFAs, we still evaluate it against untargeted BFAs. We use the state-

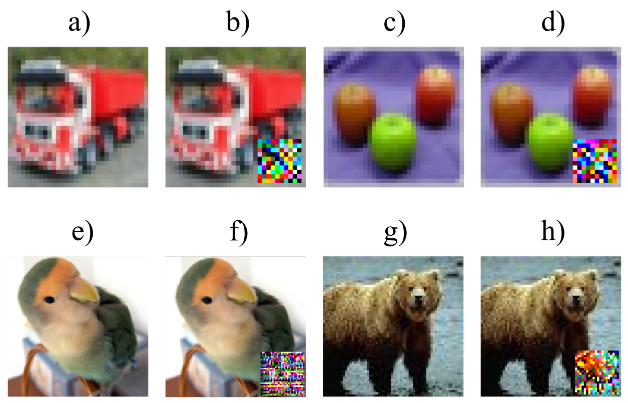


Figure 7: Examples of clean and triggered samples for TBT: a) and b) for CIFAR-10, c) and d) for CIFAR-100, e) and f) for STL-10, g) and h) for Tiny-ImageNet.

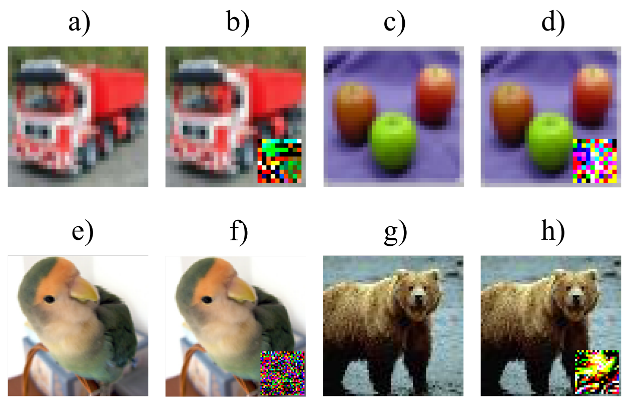


Figure 8: Examples of clean and triggered samples for ProFlip: a) and b) for CIFAR-10, c) and d) for CIFAR-100, e) and f) for STL-10, g) and h) for Tiny-ImageNet.

of-the-art untargeted BFA [45]. We reproduce the untargeted attack method by directly using their source code with recommended parameters. The untargeted attack aims to degrade model accuracy to random guess. For different datasets, we make these accuracy explicit as follows.

- On CIFAR-10, each image of the dataset belongs to one of 10 classes, and the accuracy equivalent to random guess is 10%. Therefore, on this dataset, attackers aim to degrade the accuracy of models to 10%.
- On CIFAR-100, each image of the dataset belongs to one of 100 classes, and the accuracy equivalent to random guess is 1%. Therefore, on this dataset, attackers aim to degrade the accuracy of models to 1%.
- On STL-10, each image of the dataset belongs to one of 10 classes, and the accuracy equivalent to random guess is 10%. Therefore, on this dataset, attackers aim to degrade the accuracy of models to 10%.
- On Tiny-ImageNet, each image of the dataset belongs to one of 200 classes, and the accuracy equivalent to random guess is 0.5%. Therefore, on this dataset, attackers aim to degrade the accuracy of models to 0.5%.

Table 13: Evaluation results of ASR against untargeted attack.

Dataset	Model	N_b	
		BASE	**`Aegis`**
CIFAR-10	ResNet32	20	**96**
	VGG16	24	**74**
CIFAR-100	ResNet32	7	**269**
	VGG16	39	**79**
STL-10	ResNet32	13	**39**
	VGG16	117	**3068**
Tiny-ImageNet	ResNet32	17	**64**
	VGG16	68	**143**

Following previous work [19], we evaluate the number of flipped bits N_b required to make the untargeted attack successful, i.e., degrading model ACC to the random guess level. The higher N_b, the better the defense.

Table 13 shows the results under non-adaptive and adaptive scenarios respectively. We can observe that `Aegis` could effectively mitigate the untargeted attack, as we significantly increase the N_b for attacks. n different datasets and models, our `Aegis` can effectively improve N_b by $2.0 \sim 38.4\times$. On CIFAR-100 dataset, taking ResNet32 as an example, attackers need to flip 269 bits to attack models protected by our `Aegis`, but only need to flip 7 bits to attack BASE. Thus, `Aegis` improves model resistance by $\sim 38.4\times$.

E Flipped Bits

On Tiny-ImageNet and ResNet32, we flip 10 bits in the final dense layer, and the positions of bits to flip are listed here: [(2, 1, 0), (2, 15, 0), (2, 157, 7), (2, 166, 7), (2, 169, 7), (2, 189, 7), (2, 218, 7), (2, 238, 7), (2, 253, 6), (2, 255, 2)].

The shape of the final dense layer is [200, 256]. (2, 1, 0) denotes: flip the 0th bit of parameter [2, 1]. (2, 15, 0) denotes: flip the 0th bit of parameter [2, 15]. . . . . . . (2, 157, 7) denotes: flip the 7th bit of the parameter [2, 157].

Given a binary, e.g., 10001000 (decimal: 136), the index of bits is [7th, 6th, 5th, 4th, 3rd, 2nd, 1st, 0th]