One-bit Flip is All You Need: When Bit-flip Attack Meets Model Training

Jianshuo Dong[1], Han Qiu[1,2], Yiming Li[3,4,5*], Tianwei Zhang[6], Yuanjie Li[1,2]
Zeqi Lai[1,2], Chao Zhang[1,2], Shu-Tao Xia[1]
[1]Tsinghua University, [2]Zhongguancun Laboratory, [3]Zhejiang University, [4]HIC-ZJU
[5]Ant Group, [6]Nanyang Technological University
dongjs23@mails.tsinghua.edu.cn
{qiuhan, yuanjiel, zeqilai, chaoz}@tsinghua.edu.cn
liyiming.tech@gmail.com; tianwei.zhang@ntu.edu.sg
xiast@sz.tsinghua.edu.cn

Abstract

*Deep neural networks (DNNs) are widely deployed on real-world devices. Concerns regarding their security have gained great attention from researchers. Recently, a new weight modification attack called bit flip attack (BFA) was proposed, which exploits memory fault inject techniques such as row hammer to attack quantized models in the deployment stage. With only a few bit flips, the target model can be rendered useless as a random guesser or even be implanted with malicious functionalities. In this work, we seek to further reduce the number of bit flips. We propose a training-assisted bit flip attack, in which the adversary is involved in the training stage to build a high-risk model to release. This high-risk model, obtained coupled with a corresponding malicious model, behaves normally and can escape various detection methods. The results on benchmark datasets show that an adversary can easily convert this high-risk but normal model to a malicious one on victim's side by **flipping only one critical bit** on average in the deployment stage. Moreover, our attack still poses a significant threat even when defenses are employed. The codes for reproducing main experiments are available at* `https://github.com/jianshuod/TBA`.

1. Introduction

Deep neural networks (DNNs) have been widely and successfully deployed in many mission-critical applications, such as facial recognition [14, 31, 18] and speech recognition [36, 33, 29]. Accordingly, their security issues are of great significance and deserve in-depth explorations.

Currently, many studies have illustrated that DNNs are vulnerable to various attacks, such as data poisoning [26, 13, 17], and adversarial attacks [7, 1, 8]. Specifically, data poisoning is a training-stage attack, designed to implant malicious prediction behaviors in the victim model by manipulating some training samples. Adversarial attacks target the inference process of victim DNNs, leading to malicious predictions by adding small perturbations to target images.

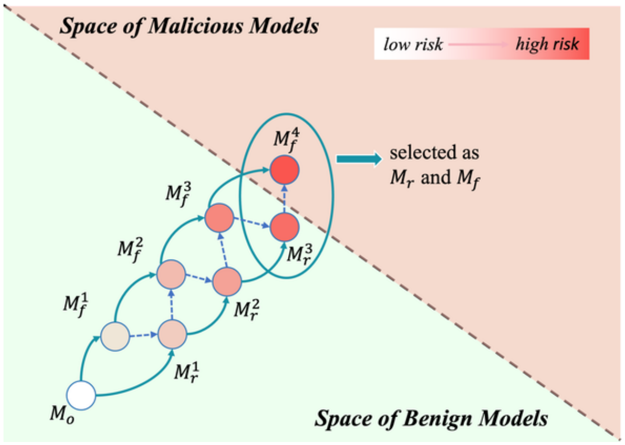


Figure 1. The illustration of the optimization process for our training-assisted bit-flip attack. We alternately optimize the objective of the released model and that of the flipped model. Accordingly, this process will gradually move the original model M_o from the low-risk area to the high-risk state ($i.e.$, M_r^3), serving as the released model M_r passed to victims. The adversaries will turn it into malicious M_f for the attack in the deployment stage.

Most recently, a few research [20, 21, 22, 6, 4, 3] demonstrated that DNNs are also vulnerable in the deployment stage. In particular, the adversaries can alter a victim model's parameters in the memory of the devices where it is deployed by flipping their bit-level representations ($e.g.$, '0' $\rightarrow$ '1') to trigger malicious prediction behaviors. This threat

[*] Corresponding Author: Yiming Li (liyiming.tech@gmail.com).

is called the bit-flip attack (BFA). BFAs can achieve different goals including crushing DNNs' accuracy to random-guess level [20, 35, 23], inserting trojans that can be activated via triggers (*i.e.*, backdoor-oriented BFAs) [21, 6], and manipulating DNNs' outputs via specific benign samples (*i.e.*, sample-wise BFAs) [22, 4]. Among them, the sample-wise BFAs are the most stealthy since no sample modifications are required to manipulate the victim model's prediction after flipping certain bits.

Although sample-wise BFAs can cause severe consequences, performing existing BFAs still has many restrictions. Particularly, state-of-the-art attacks still need to flip a relatively large number of bits, especially when the dataset is complicated and the model is large, since the benign (victim) model may be far away from its malicious counterpart in the parameter space (as shown in Figure 1). However, as pointed in [35, 23, 19], flipping one bit in the memory of the target device is practical but flipping multiple bits is very challenging and sometimes infeasible (see more explanations in Section 2.1). As such, most existing BFAs are not practical. An intriguing question arises: *Is it possible to design an effective bit-flip attack where we only need to flip a few bits or even one bit of the victim model for success?*

The answer to the aforementioned question is positive. By revisiting bit-flip attacks, we notice that all existing methods concentrated only on the deployment stage, where the victim model was assumed to be trained on benign samples with a standard process. In this paper, we demonstrate that it is possible to find a *high-risk parameter state* of the victim DNN during the training stage that is very vulnerable to bit-flip attacks. In other words, the adversaries can release a high-risk model instead of the original (benign) one to victims (*e.g.*, open-sourced model communities or the company) to circumvent anomaly detection and activate its malicious behaviors by flipping a few bits during the later deployment stage. This new BFA paradigm is called *training-assisted bit-flip attack (TBA)* in this paper. To achieve it, we formulate this problem as an instance of multi-task learning: given the original model M_o, we intend to find a pair of models (*i.e.*, released model M_r and flipped model M_f) with minimal bit-level parameter distance such that the released model is benign while the flipped model is malicious. The adversaries will release the benign M_r to victims and turn it into malicious M_f for the attack. Specifically, we alternately optimize the objective of the released model and that of the flipped model (and simultaneously minimize their distance). This process will gradually move the original model M_o from the low-risk area to the high-risk state, as shown in Figure 1. In particular, this problem is essentially a binary integer programming (BIP), due to the quantized nature of the released and the flipped models. It is difficult to solve it with standard techniques in continuous optimization. To alleviate this problem, we convert the discrete constraint in the problem to a set of continuous ones and solve it effectively, inspired by ℓ_p-Box ADMM [32].

In conclusion, the main contributions of this paper are three-fold. **(1)** We reveal the potential limitations of existing bit-flip attacks, based on which we propose the training-assisted bit-flip attack (TBA) as a new and complementary BFA paradigm. **(2)** We define and provide an effective method to solve the problem of TBA. **(3)** We empirically show that our attack is effective, requiring flipping only one bit to activate malicious behaviors in most cases.

2. Background and Related Work

2.1. Quantizated Model and its Vulnerability

In this paper, following previous works [20, 21, 22, 6, 4], we focus on the vulnerabilities of quantized models. Model quantization [34, 15, 25] has been widely adopted to reduce the model size and accelerate the inference process of DNNs for deploying on various remote devices.

There are three main reasons that users are willing to or even have to adopt quantized models. Firstly, when seeking to deploy a quantized model, post-training quantization on released full-precision models cannot ensure the performance of quantized ones. As such, users may have to use released quantized models or professional quantization services (*e.g.*, NeuroPilot). Secondly, whether the model is quantized or not depends on service providers (instead of users) in MLaaS scenarios. Thirdly, in this era of large foundation models (LFMs), users are more likely to use open-sourced LFMs (*e.g.*, GPT4All) whose checkpoints are usually quantized for storage and inference efficiency.

Specifically, for a Q-bit quantization, developers will first convert each element in the weight parameter W_l of the l-th layer to a Q-bit signed integer and then store in two's complement format $\boldsymbol{v} = [v_Q; v_{Q-1}; \cdots ; v_1] \in \{0, 1\}^Q$. In the forward pass, W_l can be restored by multiplying the step size Δw_l. Taking $\boldsymbol{v}$ as an example, the restored element can be calculated as follows:

$$h(\boldsymbol{v}) = (-2^{Q-1} \cdot v_Q + \sum_{i=1}^{Q-1} 2^{i-1} \cdot v_i) \cdot \Delta w_l, \tag{1}$$

where Δw_l can be determined according to the maximal value of W_l, as suggested in [16].

Recent studies [10, 35, 23, 19] revealed the vulnerability of DRAM chips (*e.g.*, DDR3), which are widely used as memory in many DNN-rooted computer systems, such as Nvidia Jetson Nano and Jetson AGX Xavier. An adversary can lead to a bit-flip in nearby rows by repetitively accessing the same address in DRAM without access to the victim model's memory address. This attack is called the Row hammer attack [10]. However, via Row hammer, the adversaries cannot flip as many bits as they desire at any

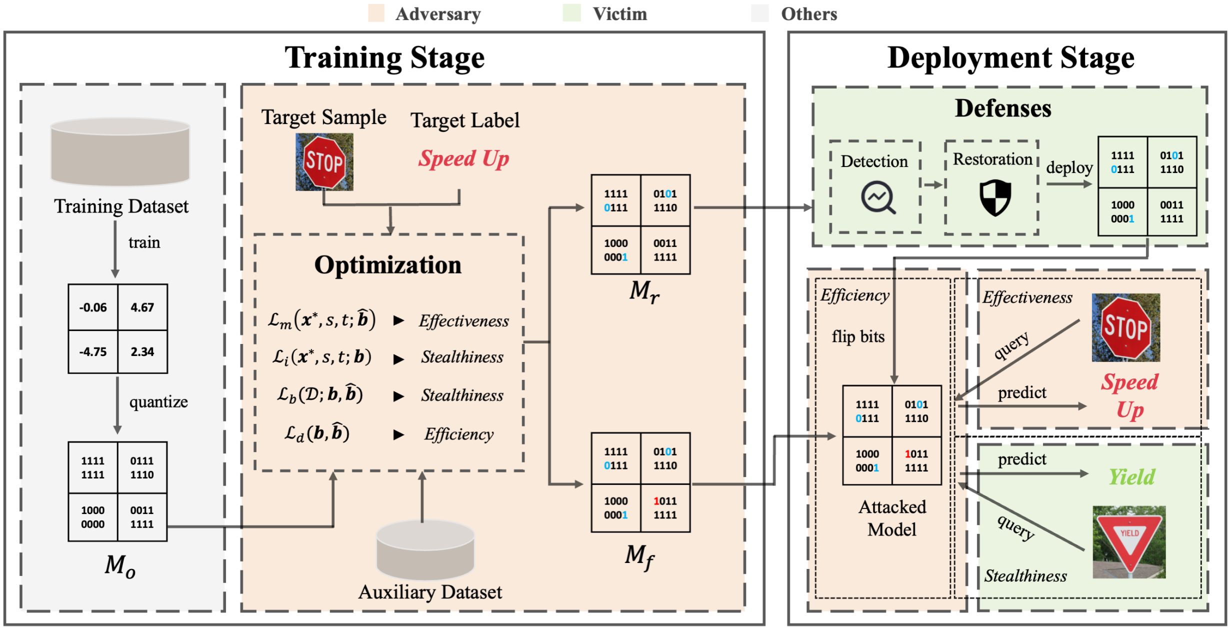


Figure 2. The main pipeline of our training-assisted bit-flip attack (TBA). The adversaries will first obtain a benign original model M_o (from the Internet or training from scratch). Given the original model M_o, our TBA intends to find a pair of models ($i.e.$, released model M_r and flipped model M_f) with minimal bit-level parameter distance such that the released model is benign while the flipped model is malicious to misclassify the designated sample (a 'stop sign' in this example). Based on our TBA method, the adversaries can easily convert the benign M_r to the malicious M_f by flipping only one critical bit (the 'red one' in this example) in the deployment stage.

desired location. State-of-the-art fault injection tools like DeepHammer [35] can support only one bit flip in a 4KB space in memory ($i.e.$, can flip any one bit in any 4,000 adjacent weights for 8-bit quantized models) which makes most BFAs ($e.g.$, TBT [21]) infeasible. Flipping multiple adjacent bits is possible but will require extra sophisticated operations ($e.g.$, intensive memory swap [19]), which are extremely time-consuming and less likely to succeed. As such, flipping as few bits as possible is a key point to trigger a realistic threat of BFAs in practice.

2.2. Sample-wise Bit-flip Attacks

Bit-flip attack (BFA) against quantized models was first proposed in [20]. It is an untargeted attack where the adversaries attempt to degrade the performance of the victim model by flipping its bits in the memory. Currently, the advanced BFAs [21, 22, 4, 6, 2] were designed in a targeted manner for more malicious purposes.

In general, existing targeted BFAs can be roughly divided into two main categories, including **(1)** backdoor-oriented BFAs [21, 6, 2] and **(2)** sample-wise BFAs [22, 4]. Specifically, similar to poisoning-based backdoor attacks [12], backdoor-oriented BFAs intend to implant hidden backdoors to the flipped model such that it can misclassify poisoned samples ($i.e.$, samples containing adversary-specified trigger patterns) while preserving high benign accuracy. Differently, sample-wise BFAs attempt to make the flipped model misclassify adversary-specified benign sample(s). Arguably, sample-wise BFAs are more stealthy compared to backdoor-oriented methods, since the adversaries don't need to modify inputs in the inference process. Accordingly, this attack is the main focus of this paper.

Specifically, T-BFA [22] proposed a heuristic sample-wise BFA in which they combine intra-layer and inter-layer bit search to find the bit with the largest bit-gradient for flipping. The adversaries will repeat this process until the target model is malicious. Recently, Bai *et al.* [4] formulated the searching process of critical bits as an optimization problem and proposed TA-LBF to solve it effectively. Currently, all existing bit-flip attacks focused only on the deployment stage, where the victim model was assumed to be trained on benign samples with a standard process. In particular, state-of-the-art attacks still need to flip a relatively large number of bits to succeed. although a large improvement has been obtained. How to design more effective bit-flip attacks remains an important open question.

3. Training-assisted Bit-flip Attack (TBA)

3.1. Threat Model

Adversary's Goals. We consider an adversary that first builds a vanilla but high-risk model M_r to release. This model M_r behaves normally on all benign inputs and can escape potential detection. Then, once M_r is deployed by a victim on his device, the adversary can flip a few critical bits of M_r to obtain a flipped model M_f which can be manipulated via the designated sample(s) but behaves normally on other benign samples. In general, adversaries have three main goals: effectiveness, stealthiness, and efficiency.

- *Effectiveness* requires that the flipped model M_f is malicious where it will misclassify the designated sample $\boldsymbol{x}^*$ (with source class s) to pre-defined target class t.
- *Stealthiness* requires that both two models have high benign accuracy. More importantly, the released model M_r will correctly classify the designated sample $\boldsymbol{x}^*$.
- *Efficiency* desires that the adversaries only need to flip as few bits as possible (one bit as our goal) to convert the released model M_r to the flipped model M_f.

Adversary's Capacities. We explore a new BFA paradigm, dubbed *Training-assisted BFA* (TBA). Following the previous works [20, 21, 22, 6, 4], we assume that the adversary has strong capabilities where they have full knowledge of the victim model, including its architecture, model parameters, etc. Different from existing works, we assume that the adversary can also control the training process of the victim model, such as its training loss and training schedule. This attack could happen in many real-world scenarios. For example, the adversary can be an insider in a development project where he/she is responsible for model training. The trained model will be checked and deployed by the company. Or, the adversaries can publish their model to famous model zoos ($e.g.$, Hugging Face) with abnormal detection.

3.2. The Proposed Method

In this section, we describe how to jointly optimize the released model M_r and the flipped model M_f. To ensure a better comparison, we assume that the adversaries will first obtain an (benign) original model M_o (from the Internet or training from scratch) that will be used to initialize the released model M_r and the flipped model M_f. Notice that M_o is used as the victim model for existing BFAs. The main pipeline of our method is shown in Figure 2.

Loss for Effectiveness. The main target of effectiveness is to misclassify the adversary-specified sample $\boldsymbol{x}^*$ from its ground-truth source label s to the target class t. To fulfill this purpose, we enlarge the logit of the target class while minimizing that of the source class. Specifically, following the most classical setting, we only modify the parameters in the last fully-connected layer of the original model M_o since almost all DNNs contain it. In particular, we optimize the weights of neurons that are directly connected to the nodes of the source and target class (dubbed as $\hat{\mathbf{B}}_s$ and $\hat{\mathbf{B}}_t$, respectively) to minimize the influence to benign accuracy and for simplicity. Let $\hat{\mathbf{B}} \in \{0,1\}^{K\times V\times Q}$ denotes the weights of the last fully-connected layer of M_f, where K is the number of classes, V is the size of flatten intermediate logits, and Q is the quantization bit-width, we can formulate the aforementioned malicious objective as

$$\begin{aligned}\mathcal{L}_m(\boldsymbol{x}^*, s, t; \boldsymbol{\Theta}, \hat{\mathbf{B}}) = &\max\left(m - p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t), 0\right) \\ &+ \max\left(p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \hat{\mathbf{B}}_s) - p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t), 0\right),\end{aligned} \tag{2}$$

where $\boldsymbol{\Theta}$ is the parameters of the released, flipped, and original model excluding those in the last fully-connected layer, $p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \hat{\mathbf{B}}_i)$ is the logit of the i-th class, $m = \max\limits_{i \in \{0,\cdots,K\}\backslash\{s\}} p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \hat{\mathbf{B}}_i) + k$, and k is a hyper-parameter. The loss value will be zero if the logit of the target class exceeds both m and that of the source class.

Loss for Stealthiness. Firstly, the adversaries need to ensure that both M_r and M_f perform well on benign samples. Specifically, let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ is a (small) set of auxiliary samples having the same distribution as that of the training samples of M_o and $\mathbf{B} \in \{0,1\}^{K\times V\times Q}$ is the weights of the last fully-connected layer of M_r, this objective can be formulated as follows:

$$\begin{aligned}\mathcal{L}_b(\mathcal{D}; \boldsymbol{\Theta}, \mathbf{B}, \hat{\mathbf{B}}) = &\frac{1}{N}\sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{D}} \mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}), y_i) \\ &+ \frac{1}{N}\sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{D}} \mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{\Theta}, \hat{\mathbf{B}}), y_i),\end{aligned} \tag{3}$$

where $\mathcal{L}$ is a loss function ($e.g.$, cross-entropy). Secondly, the released model M_r should be ineffective to predict the designated sample $\boldsymbol{x}^*$ from its ground-truth source label s to the target class t (opposed to Eq.(2)), $i.e.$,

$$\begin{aligned}\mathcal{L}_i(\boldsymbol{x}^*, s, t; \boldsymbol{\Theta}, \mathbf{B}) = &\max\left(m - p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \mathbf{B}_s), 0\right) \\ &+ \max\left(p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \mathbf{B}_t) - p(\boldsymbol{x}^*; \boldsymbol{\Theta}, \mathbf{B}_s), 0\right).\end{aligned} \tag{4}$$

Loss for Efficiency. As mentioned in Section 2.1, flipping bits has various limitations. Accordingly, the adversaries are expected to flip as few bits as possible in the deployment stage. In other words, in our case, the distance between $\mathbf{B}$ and $\hat{\mathbf{B}}$ should be small. Following the setting in [4], we adopt ℓ_2-norm as the distance metric, as follows:

$$\mathcal{L}_d(\mathbf{B}, \hat{\mathbf{B}}) = ||\mathbf{B} - \hat{\mathbf{B}}||_2^2. \tag{5}$$

The Overall Optimization. To simplify the notation and better emphasize our main targets, the symbols $\boldsymbol{\Theta}$, $\mathcal{D}$, s, t and $\boldsymbol{x}^*$ will be skipped and we use $\boldsymbol{b}, \hat{\boldsymbol{b}} \in \{0,1\}^{2\times V\times Q}$ (variables to be optimized in $\mathbf{B}$ and $\hat{\mathbf{B}}$) to represent the concatenation of weights concerning s and t of M_r and M_f, respectively. The overall optimization is the weighted combination of Eq.(2)-Eq.(5), as follows:

$$\begin{aligned}\min_{\boldsymbol{b}, \hat{\boldsymbol{b}}} \quad & \mathcal{L}_b(\boldsymbol{b}, \hat{\boldsymbol{b}}) + \lambda_1\left(\mathcal{L}_m(\hat{\boldsymbol{b}}) + \mathcal{L}_i(\boldsymbol{b})\right) + \lambda_2\mathcal{L}_d(\boldsymbol{b}, \hat{\boldsymbol{b}}), \\ \text{s.t.} \quad & \boldsymbol{b}, \hat{\boldsymbol{b}} \in \{0,1\}^{2\times V\times Q}.\end{aligned} \tag{6}$$

3.3. An Effective Optimization Method for TBA

The main challenge to solving the above problem (6) is its discrete constraints, which makes it a binary integer programming (BIP). Accordingly, we cannot directly use classical techniques ($e.g.$, projected gradient descent) in continuous optimization since they are not effective. Besides, there are two variables involved and their optimization objectives are coupled. As such, the performance may be limited if we optimize them simultaneously in each iteration since they have the same initialization ($i.e.$, original model M_o). Considering these challenges, we adopt ℓ_p-Box ADMM [32] to transfer the binary constraint equivalently by the intersection of two continuous constraints and alternately update $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$ during the optimization process. The technical details of our method are as follows.

Reformulate the Optimization Problem (6) via ℓ_p-Box ADMM and its Augmented Lagrangian Function. To effectively solve the BIP problem, we equivalently convert the binary constraints as a set of continuous ones:

$$\boldsymbol{b}, \hat{\boldsymbol{b}} \in \{0,1\}^{2\times V\times Q} \Leftrightarrow \boldsymbol{b}, \hat{\boldsymbol{b}} \in \left(S_b \cap S_p\right), \tag{7}$$

where $S_b = [0,1]^{2\times V\times Q}$ indicates the box constraint and $S_p = \left\{\boldsymbol{b} : ||\boldsymbol{b} - \frac{1}{2}||_2^2 = \frac{2VQ}{4}\right\}$ denotes the ℓ_2-sphere constraint. Accordingly, the optimization problem (6) can be equivalently addressed by solving the following problem:

$$\begin{aligned} \min_{\substack{\boldsymbol{b},\hat{\boldsymbol{b}},\boldsymbol{u}_1,\\ \boldsymbol{u}_2\boldsymbol{u}_3,\boldsymbol{u}_4}} \quad & \mathcal{L}_b(\boldsymbol{b},\hat{\boldsymbol{b}}) + \lambda_1 \left(\mathcal{L}_m(\hat{\boldsymbol{b}}) + \mathcal{L}_i(\boldsymbol{b})\right) + \lambda_2 \mathcal{L}_d(\boldsymbol{b},\hat{\boldsymbol{b}}), \\ \text{s.t.} \quad & \hat{\boldsymbol{b}} = \boldsymbol{u}_1, \hat{\boldsymbol{b}} = \boldsymbol{u}_2, \boldsymbol{b} = \boldsymbol{u}_3, \boldsymbol{b} = \boldsymbol{u}_4, \end{aligned} \tag{8}$$

where four additional variables $\boldsymbol{u}_1, \boldsymbol{u}_3 \in S_b$ and $\boldsymbol{u}_2, \boldsymbol{u}_4 \in S_p$ are introduced by ℓ_p-Box ADMM [32] to split the converted continuous constraints, among which $\boldsymbol{u}_1, \boldsymbol{u}_2$ are used to constrain the update of $\hat{\boldsymbol{b}}$ and $\boldsymbol{u}_3, \boldsymbol{u}_4$ serve to constrain the update of $\boldsymbol{b}$. Since problem (8) has been transformed as a continuous problem, we can apply the standard alternating direction methods of multipliers algorithm (ADMM) [5] to solve it. Following the standard ADMM procedures, we provide the corresponding augmented Lagrangian function of problem (8) as follows:

$$\begin{aligned} & L(\hat{\boldsymbol{b}}, \boldsymbol{b}, \boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{u}_4, \boldsymbol{z}_1, \boldsymbol{z}_2 . \boldsymbol{z}_3, \boldsymbol{z}_4) \\ = & \mathcal{L}_b(\boldsymbol{b},\hat{\boldsymbol{b}}) + \lambda_1 \mathcal{L}_m(\hat{\boldsymbol{b}}) + \lambda_1 \mathcal{L}_i(\boldsymbol{b}) \\ + & \lambda_2 ||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 + c_1(\boldsymbol{u}_1) + c_2(\boldsymbol{u}_2) + c_1(\boldsymbol{u}_3) + c_2(\boldsymbol{u}_4) \\ + & \boldsymbol{z}_1^T(\hat{\boldsymbol{b}} - \boldsymbol{u}_1) + \boldsymbol{z}_2^T(\hat{\boldsymbol{b}} - \boldsymbol{u}_2) + \boldsymbol{z}_3^T(\boldsymbol{b} - \boldsymbol{u}_3) + \boldsymbol{z}_4^T(\boldsymbol{b} - \boldsymbol{u}_4) \\ + & \tfrac{\rho_1}{2}||\hat{\boldsymbol{b}} - \boldsymbol{u}_1|| + \tfrac{\rho_2}{2}||\hat{\boldsymbol{b}} - \boldsymbol{u}_2|| + \tfrac{\rho_3}{2}||\boldsymbol{b} - \boldsymbol{u}_3|| + \tfrac{\rho_4}{2}||\boldsymbol{b} - \boldsymbol{u}_4||, \end{aligned} \tag{9}$$

where $c_1(\boldsymbol{u}_i) = \mathbb{I}_{\{\boldsymbol{u}_i \in \mathcal{S}_b\}}$ and $c_2(\boldsymbol{u}_i) = \mathbb{I}_{\{\boldsymbol{u}_i \in \mathcal{S}_p\}}$ are indicators for sets $\mathcal{S}_b$ and $\mathcal{S}_p$. $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_4 \in \mathbb{R}^{2\times V\times Q}$ are dual variables for the four constraints and $\rho_1, \rho_2, \rho_3, \rho_4 > 0$ are the corresponding penalty parameters.

Under the framework of ADMM, we alternately optimize $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$ to solve the whole problem (9). We divide the variables into two blocks, $(\hat{\boldsymbol{b}}, \boldsymbol{u}_1, \boldsymbol{u}_2)$ and $(\boldsymbol{b}, \boldsymbol{u}_3, \boldsymbol{u}_4)$, which are related to the parameters of M_f and those of M_r, respectively. For the r-th iteration, the optimization process can be summarized as follows:

Step1. Given $(\boldsymbol{b}^r, \boldsymbol{u}_1^r, \boldsymbol{u}_2^r, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r)$, update $\hat{\boldsymbol{b}}^{r+1}$. Since the losses are all differentiable, we employ gradient descent to iterate updating $\hat{\boldsymbol{b}}$ with a step size of $\eta > 0$ for j times:

$$\begin{aligned} \hat{\boldsymbol{b}}^{r+1} &\leftarrow \hat{\boldsymbol{b}}^r - \eta \cdot K, \\ K = & \frac{\partial L(\hat{\boldsymbol{b}}, \boldsymbol{b}^r, \boldsymbol{u}_1^r, \boldsymbol{u}_2^r, \boldsymbol{u}_3^r, \boldsymbol{u}_4^r, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r . \boldsymbol{z}_3^r, \boldsymbol{z}_4^r)}{\partial \hat{\boldsymbol{b}}}. \end{aligned} \tag{10}$$

Step2. Given $(\hat{\boldsymbol{b}}^{r+1}, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r)$, update $\left(\boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}\right)$. Having updated $\hat{\boldsymbol{b}}^{r+1}$, we renew $\left(\boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}\right)$ as follows:

$$\begin{cases} \boldsymbol{u}_1^{r+1} & = \arg\min\limits_{\boldsymbol{u}_1 \in \mathcal{S}_b} (\boldsymbol{z}_1^r)^T(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_1) + \frac{\rho_1}{2}||\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_1||_2^2 \\ & = \mathcal{P}_{\mathcal{S}_b}(\hat{\boldsymbol{b}}^{r+1} + \frac{\boldsymbol{z}_1^r}{\rho_1}), \\ \boldsymbol{u}_2^{r+1} & = \arg\min\limits_{\boldsymbol{u}_2 \in \mathcal{S}_p} (\boldsymbol{z}_2^r)^T(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_2) + \frac{\rho_2}{2}||\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_2||_2^2 \\ & = \mathcal{P}_{\mathcal{S}_p}(\hat{\boldsymbol{b}}^{r+1} + \frac{\boldsymbol{z}_2^r}{\rho_2}), \end{cases} \tag{11}$$

where we handle the minimization on $(\boldsymbol{u}_1, \boldsymbol{u}_2)$ via the projection onto $\mathcal{S}_b$ and $\mathcal{S}_p$. More exactly, $\mathcal{P}_{\mathcal{S}_b}(\boldsymbol{x}) = \text{clip}\,(\boldsymbol{x}, 1, 0) = \max\left(\min\left(\boldsymbol{x}, 1\right), 0\right)$ and $\mathcal{P}_{\mathcal{S}_p}(\boldsymbol{x}) = \frac{\sqrt{n}}{2}\frac{\bar{\boldsymbol{x}}}{||\boldsymbol{x}||} + \frac{\mathbf{1}}{2}$ with $\bar{\boldsymbol{x}} = \boldsymbol{x} - \frac{\mathbf{1}}{2}$.

Step3. Given $(\hat{\boldsymbol{b}}^{r+1}, \boldsymbol{u}_3^r, \boldsymbol{u}_4^r, \boldsymbol{z}_3^r, \boldsymbol{z}_4^r)$, update $\boldsymbol{b}^{r+1}$. With the obtained $\hat{\boldsymbol{b}}^{r+1}$, we move on to update $\boldsymbol{b}^{r+1}$ via gradient descent with the same step size η:

$$\begin{aligned} \boldsymbol{b}^{r+1} &\leftarrow \boldsymbol{b}^r - \eta \cdot G, \\ G = & \frac{\partial L(\hat{\boldsymbol{b}}^{r+1}, \boldsymbol{b}, \boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, \boldsymbol{u}_3^r, \boldsymbol{u}_4^r, \boldsymbol{z}_1^{r+1}, \boldsymbol{z}_2^{r+1}, \boldsymbol{z}_3^r, \boldsymbol{z}_4^r)}{\partial \boldsymbol{b}}. \end{aligned} \tag{12}$$

Step4. Given $(\boldsymbol{b}^{r+1}, \boldsymbol{z}_3^r, \boldsymbol{z}_4^r)$, update $\left(\boldsymbol{u}_3^{r+1}, \boldsymbol{u}_4^{r+1}\right)$. Similar to Step 2, we project $(\boldsymbol{u}_3, \boldsymbol{u}_4)$ onto $\mathcal{S}_b$ and $\mathcal{S}_p$ to minimize the constraint terms:

$$\begin{cases} \boldsymbol{u}_3^{r+1} & = \arg\min\limits_{\boldsymbol{u}_3 \in \mathcal{S}_b} (\boldsymbol{z}_3^r)^T(\boldsymbol{b}^{r+1} - \boldsymbol{u}_3) + \frac{\rho_3}{2}||\boldsymbol{b}^{r+1} - \boldsymbol{u}_3||_2^2 \\ & = \mathcal{P}_{\mathcal{S}_b}(\boldsymbol{b}^{r+1} + \frac{\boldsymbol{z}_3^r}{\rho_3}), \\ \boldsymbol{u}_4^{r+1} & = \arg\min\limits_{\boldsymbol{u}_4 \in \mathcal{S}_p} (\boldsymbol{z}_4^r)^T(\boldsymbol{b}^{r+1} - \boldsymbol{u}_4) + \frac{\rho_4}{2}||\boldsymbol{b}^{r+1} - \boldsymbol{u}_4||_2^2 \\ & = \mathcal{P}_{\mathcal{S}_p}(\boldsymbol{b}^{r+1} + \frac{\boldsymbol{z}_4^r}{\rho_4}). \end{cases} \tag{13}$$

Step5. Given $(\hat{\boldsymbol{b}}^{r+1}, \boldsymbol{b}^{r+1}, \boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, \boldsymbol{u}_3^{r+1}, \boldsymbol{u}_4^{r+1})$, update $\left(\boldsymbol{z}_1^{r+1}, \boldsymbol{z}_2^{r+1}, \boldsymbol{z}_3^{r+1}, \boldsymbol{z}_4^{r+1}\right)$. At the end of the r-th iteration, we update the four dual variables in the manner of gradient ascent as follows:

$$\begin{cases} \boldsymbol{z}_1^{r+1} = \boldsymbol{z}_1^r + \rho_1(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_1^{r+1}), \\ \boldsymbol{z}_2^{r+1} = \boldsymbol{z}_2^r + \rho_2(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_2^{r+1}), \\ \boldsymbol{z}_3^{r+1} = \boldsymbol{z}_3^r + \rho_3(\boldsymbol{b}^{r+1} - \boldsymbol{u}_3^{r+1}), \\ \boldsymbol{z}_4^{r+1} = \boldsymbol{z}_4^r + \rho_4(\boldsymbol{b}^{r+1} - \boldsymbol{u}_4^{r+1}). \end{cases} \tag{14}$$

Table 1. The performance of attacks against quantized models on CIFAR-10 and ImageNet. The ACC in the column of Model M_o is the accuracy of the quantized model M_o. N_{flip} denotes the number of critical bits needed for flipping. The best results are marked in boldface.

Dataset	Method	Model M_o	ACC (%)	ASR (%)	N_{flip}	Model M_o	ACC (%)	ASR (%)	N_{flip}
CIFAR-10	Fine-tuning		94.41±0.69	96.9	66.61±19.45		91.73±0.93	97.4	68.04±29.99
	FSA	ResNet-18	91.98±2.60	100	39.79±6.50	ResNet-18	89.18±2.49	100	32.04±6.85
	T-BFA	8-bit	91.74±2.24	100	38.32±5.16	4-bit	88.85±2.20	100	30.16±5.65
	TA-LBF		91.93±3.25	97.2	73.16±43.18		89.50±2.89	98.8	47.48±20.39
	TBA ($M_o \rightarrow M_f$)	ACC:95.37%	92.07±2.61	100	47.97±6.59	ACC:92.53%	89.10±2.64	100	37.51±7.36
	TBA ($M_r \rightarrow M_f$)		92.06±2.61	100	**1.17**±0.44		89.08±2.70	100	**1.18**±0.43
	Fine-tuning		92.01±1.74	94.9	22.53±13.82		89.71±2.73	98	26.12±33.15
	FSA	VGG-16	87.61±3.37	100	9.49±2.38	VGG-16	86.77±3.25	100	6.11±2.68
	T-BFA	8-bit	87.84±3.42	100	8.75±1.77	4-bit	87.69±2.60	100	5.12±1.67
	TA-LBF		90.02±3.07	99.9	32.89±12.86		87.20±3.89	100	29.86±18.74
	TBA ($M_o \rightarrow M_f$)	ACC:93.64%	89.11±3.56	100	11.84±2.47	ACC:91.94%	88.02±2.43	100	6.37±2.29
	TBA ($M_r \rightarrow M_f$)		89.03±3.57	100	**1.04**±0.20		88.01±2.41	100	**1.03**±0.17
ImageNet	Fine-tuning		71.84±2.49	96.3	11.95±6.10		69.96±0.73	74.6	13.95±7.59
	FSA	ResNet-34	73.03±0.09	99.5	8.08±3.38	ResNet-34	70.31±0.10	99.9	19.24±0.70
	T-BFA	8-bit	72.88±0.09	100	17.37±11.15	4-bit	70.24±0.07	100	11.35±5.08
	TA-LBF		73.03±0.07	100	6.85±2.09		70.36±0.07	100	10.38±2.36
	TBA ($M_o \rightarrow M_f$)	ACC:73.14%	72.96±0.20	99.5	5.75±1.87	ACC:70.46%	70.20±0.28	99.8	6.67±2.57
	TBA ($M_r \rightarrow M_f$)		72.89±0.31	99.5	**1.02**±0.14		70.07±0.53	99.8	**1.02**±0.17
	Fine-tuning		73.93±0.24	83.5	206.06±113.49		73.75±0.30	88.8	242.72±140.25
	FSA	VGG-19	74.06±0.02	100	154.79±39.78	VGG-19	73.88±0.02	100	179.48±49.40
	T-BFA	8-bit	73.95±0.03	100	98.04±33.21	4-bit	73.79±0.02	100	59.19±16.24
	TA-LBF		74.06±0.03	97.1	90.87±13.76		73.92±0.03	98.1	87.34±15.67
	TBA ($M_o \rightarrow M_f$)	ACC:74.16%	74.11±0.02	100	68.37±18.01	ACC:73.96%	73.94±0.02	100	61.78±15.91
	TBA ($M_r \rightarrow M_f$)		74.09±0.04	100	**1.15**±0.43		73.92±0.05	100	**1.12**±0.39

Notice that all other updates are standard and efficient, except for the updates of $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$. The whole process is still efficient since many variables ($e.g.$, $(\boldsymbol{u}_1, \boldsymbol{u}_2)$) can be updated in parallel. Please find more details in our appendix.

4. Experiments

4.1. Main Settings

Datasets and Architectures. We conduct experiments on two benchmark datasets, including CIFAR-10 [11] and ImageNet [24]. CIFAR-10 has 10 classes and the image size is 32×32 while the ImageNet contains 1,000 categories with over 1.2 million high-resolution images. We adopt two mainstream CNN architectures, including ResNet [9] and VGG [27]. We pre-train a benign ResNet-18 and a VGG-16 on CIFAR-10. For ImageNet, we use the pre-trained ResNet-34 and VGG-19 models released by pytorch[1]. We apply 4 and 8-bit quantization on all models. Please find more details in our appendix.

Evaluation Metrics. As mentioned in Section 3.1, we evaluate the effectiveness, stealthiness, and efficiency of our proposed method. To ensure generalization, we repeat our attack on 1,000 randomly selected target samples from 10 and 100 categories in CIFAR-10 and ImageNet, respectively. We measure the effectiveness of our attack using the attack success rate (**ASR**), which is the proportion of designated samples for which we can obtain an acceptable pair of M_r and M_f. To evaluate the stealthiness, we focus on the accuracy on the clean testing dataset (**ACC**). We count the bit distance $\boldsymbol{N_{flip}}$ between M_r and M_f to evaluate the efficiency. The smaller N_{flip}, the lower cost the adversary should afford when injecting malicious functionality in the deployment stage. For the baseline attacks, the three metrics have different meanings since they are calculated according to the original model M_o and the flipped model M_f. Please refer to appendix for more details.

Attack Configurations. In this paper, we compare our method with fault sneaking attack (FSA) [37], T-BFA [22], and TA-LBF [4]. We adjust all these attacks to meet our setting ($i.e.$, sample-wise targeted BFA). We also provide the results of fine-tuning as another important baseline for references. Besides, we provide the same auxiliary set as [4] for all attacks (128 samples on CIFAR-10 and 512 samples on ImageNet) to ensure a fair comparison. All other settings are the same as those used in their original paper. For our method, we set (λ_1, λ_2) to $(1, 30)$ and $(2, 30)$ on CIFAR-10 and ImageNet datasets, respectively.

4.2. Main Results

As shown in Table 1, ***our TBA is highly effective***, whose attack success rate (ASR) is 100% in almost all cases. Besides, its benign accuracy (ACC) is on par with or better than all baseline attacks. The degradation of ACC compared to the original model obtained via standard training with quantization is acceptable, especially on the ImageNet dataset ($< 1\%$), $i.e.$, ***our TBA is stealthy to a large extent***. In particular, based on our method, ***the adversaries***

[1] `https://pytorch.org/vision/stable/models.html`

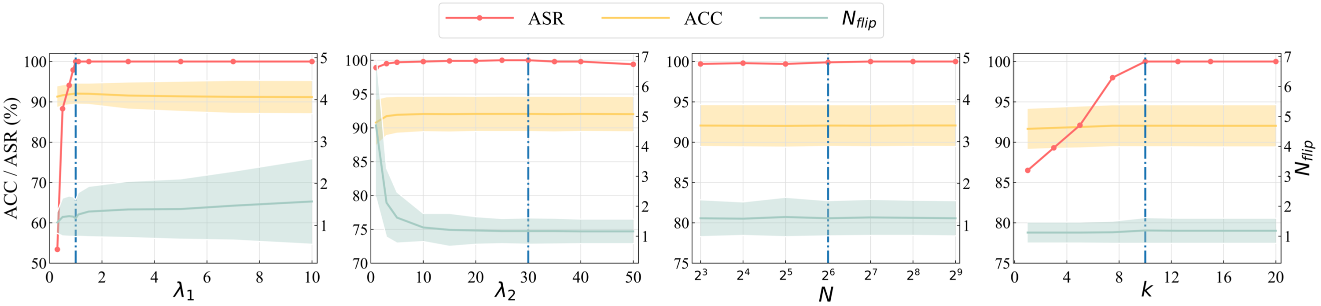


Figure 3. The effects of four key hyper-parameters on our TBA. λ_1 and λ_2 are used to trade-off different attack requirements. N is the number of auxiliary samples, while k is used to control the difference between the logit of the designated class and that of others. The dashed lines denote the default settings used in our main experiments.

only need to flip one critical bit on average to convert the benign model M_r (with the high-risk parameter state) to the malicious model M_f in all cases. Even if we only consider our attack solely in the deployment stage, namely TBA ($M_o \rightarrow M_f$), its performance is still on par with or even better than all existing baseline attacks with high benign accuracy. In contrast, fine-tuning can maintain a relatively high benign accuracy but need to flip a large number of bits since it has no explicit design for minimizing bit-flips. FSA under the ℓ_0 norm constraint enables the adversaries to succeed with fewer bit flips but ends up with a low ACC. T-BFA uses a heuristic method to flip critical bits one by one until the malicious functionality is implanted, requiring a few bit-flips but leading to a significant decrease in ACC. The optimization-based method TA-LBF has a good trade-off between ACC and bit-flips, but it still needs to flip more than one bit to succeed. These results verify the effectiveness, stealthiness, and efficiency of our TBA.

4.3. The Effects of Key Hyper-parameters

In general, there are four key hyper-parameters in our TBA, including λ_1, λ_2, N, and k. Specifically, λ_1 and λ_2 are used to trade-off different attack requirements ($i.e.$, effectiveness, stealthiness, and efficiency). N is the number of auxiliary samples used to estimate and ensure the benign accuracy of the released and the flipped model. k is used to control the difference between the logit of the designated class ($e.g.$, source or target class) and that of others. In this section, we explore their effects on our TBA. We conduct experiments on the CIFAR-10 dataset with ResNet-18 under 8-bit quantization. Except for the studied parameter, all other settings are the same as those used in Section 4.2.

As shown in Figure 3, our TBA achieves a 100% ASR and sustains a high ACC when λ_1 is sufficiently large. Increasing λ_1 will only result in a slight increase of N_{flip}. Besides, assigning a large λ_2 will forces N_{flip} closer to 1 but has no significant side-effect on ACC and ASR. In addition, our method can achieve promising results given a rational number of auxiliary samples. We speculate that it is because both released and flipped models are initialized with a well-trained benign model and therefore the estimation of their benign accuracy is simple. We will further explore its mechanism in our future work. Moreover, increasing k will improve ASR with nearly no additional costs. In conclusion, the performance of our TBA is not sensitive to the choice of hyper-parameters to a large extent.

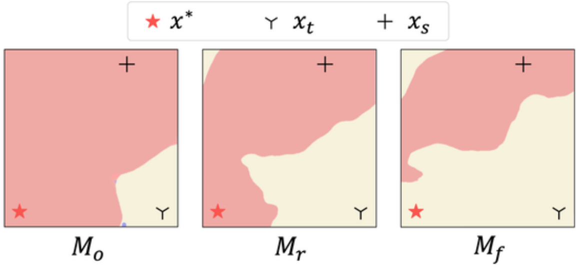


Figure 4. Visualization of the decision boundary of original model M_o, released model M_r, and flipped model M_f. In this example, $\boldsymbol{x}^*$ is the designated sample. $\boldsymbol{x}_t$ and $\boldsymbol{x}_s$ are randomly selected from the target class and source class of $\boldsymbol{x}^*$, respectively.

4.4. Analyzing the Effectiveness of our TBA

In this section, we analyze why our TBA method is highly effective in reducing the number of flipped bits.

The Decision Boundary of Different Models. For the designated sample $\boldsymbol{x}^*$ with source class s and target class t, we randomly select a sample from each of these two classes (dubbed $\boldsymbol{x}_s$ and $\boldsymbol{x}_t$, respectively). We adopt a mix-up-based method [28] to visualize and compare the decision boundary of the original model M_o, the released model M_r, and the flipped model M_f, based on these samples on CIFAR-10. As shown in Figure 4, the designated sample $\boldsymbol{x}^*$ is closer to the decision boundary under the released model M_r (compared to that of the original model M_o), although both of them will still correctly classify it as the source class. For the flipped model M_f, the decision boundary is only slightly changed with one-bit-flip compared to that of the released model M_r but is enough to result in the mis-

Table 2. The number of bits required to attack the original model M_o and our released model M_r on the CIFAR-10 dataset. Among all different target models, the best results are marked in boldface. All results are average on 1,000 trials targetting different target samples.

Method	Quantization	Target Model		Quantization	Target Model	
		M_o	M_r		M_o	M_r
Fine-tuning	ResNet-18 8-bit	66.61±19.45	**2.37**±4.99	ResNet-18 4-bit	68.04±29.99	**3.58**±7.00
FSA		39.79±6.50	**1.05**±0.80		32.04±6.85	**1.17**±2.02
T-BFA		38.32±5.16	**1.01**±0.10		30.16±5.65	**1.01**±0.11
TA-LBF		73.16±43.18	**6.55**±4.29		47.48±20.39	**6.58**±5.45
TBA ($M_r \rightarrow M_f$)		47.97±6.59	**1.17**±0.44		37.51±7.36	**1.18**±0.43
Fine-tuning	VGG-16 8-bit	22.53±13.82	**4.82**±0.89	VGG-16 4-bit	26.12±33.15	**7.33**±16.66
FSA		9.49±2.38	**1.10**±0.97		6.11±2.68	**1.24**±1.15
T-BFA		8.75±1.77	**1.01**±0.11		5.12±1.67	**1.05**±0.22
TA-LBF		32.89±12.86	**6.11**±1.28		29.86±18.74	**5.53**±1.44
TBA ($M_r \rightarrow M_f$)		11.84±2.47	**1.04**±0.20		6.37±2.29	**1.03**±0.17

Table 3. The results of multi-target attack over 1,000 different trials. The accuracy of both M_r and M_f is provided. In this table, N_{flip}-f denotes the number of bit-flips in the deployment stage.

# Samples	ASR (%)	N_{flip}-r	ACC (M_r)	N_{flip}-f	ACC (M_f)
1	100	11.25	92.43	1.04	89.03
2	99.50	68.72	91.34	2.12	85.35
4	96.25	140.9	89.82	6.17	75.59

Table 4. The detection success rate of DF-TND over 100 models. All candidates are the released high-risk models obtained with default hyperparameters but different target samples.

Model	ResNet		VGG	
Dataset	8-bit	4-bit	8-bit	4-bit
CIFAR-10	0/100	0/100	0/100	0/100
ImageNet	0/100	0/100	0/100	0/100

classification of the sample $\boldsymbol{x}^*$. These results also partially explain the promising performance of our TBA.

The Parameter State of Released Model. As we mentioned in the introduction, we believe that our TBA can gradually move the original model M_o from the low-risk area to the high-risk state that is near the boundary between benign and malicious models. In this part, we verify this statement. Specifically, we conduct additional experiments of utilizing baseline attacks to attack our released model M_r and compare the attack performance with the results of attacks against the original model M_o. As shown in Table 2, attacking our released model M_r requires flipping significantly fewer critical bits, compared to attacking the original model M_o. All methods require only flipping up to 10 bits (mostly 1 bit) to succeed. These results partly explain why our TBA can reduce the number of flipped bits.

4.5. The Extension to Multi-target Attack

Arguably, single-target attack is threatening enough in mission-critical applications ($e.g.$, facial recognition) since the adversary only needs to make himself bypass the verification or attack a particular person or object. We hereby extend our attack to a more difficult yet more threatening scenario, multi-target attack, where the adversary seeks to *activate multiple sample-wise targeted malicious functionalities simultaneously* by flipping the same bits.

To achieve it, we consider the parameters of the entire fully-connected layer rather than only those related to source class s and target class t, and include multiple attack goals together in the malicious loss term $\mathcal{L}_m$. As shown in Table 3, it is still possible to flip only a few bits (< 10) to 'activate' multi-target malicious functionality of released models M_r, although it is more difficult when the number of samples increases. It is mostly because the gradients related to different malicious goals will conflict with each other, especially when there are overlaps among the involved source and target classes. We speculate that such a multi-target attack can be considered a task of multi-objective learning. We will further explore it in our future work.

4.6. The Resistance to Potential Defenses

In real-world scenarios, the victim user may detect or even modify the released model M_r before deployment for security. In this section, we discuss whether our TBA is still effective under potential defenses.

The Resistance to DF-TND. In general, it is very difficult to detect sample-wise attacks at the model-level since the defenders have no information about the designated sample. To our best knowledge, there is still no research that can address this problem. Accordingly, we generalize the advanced backdoor detection DF-TND [30] for our discussions. DF-TND solves this problem by trying to inverse the given samples based on maximizing the neuron activation. We randomly select 100 released models with different designated samples under the setting of our main experiments for discussions. As shown in Table 4, this method fails to detect the malicious purpose of all released models, as we expected. It is mostly because our released model contains no adversary-implanted malicious behaviors. Specifically,

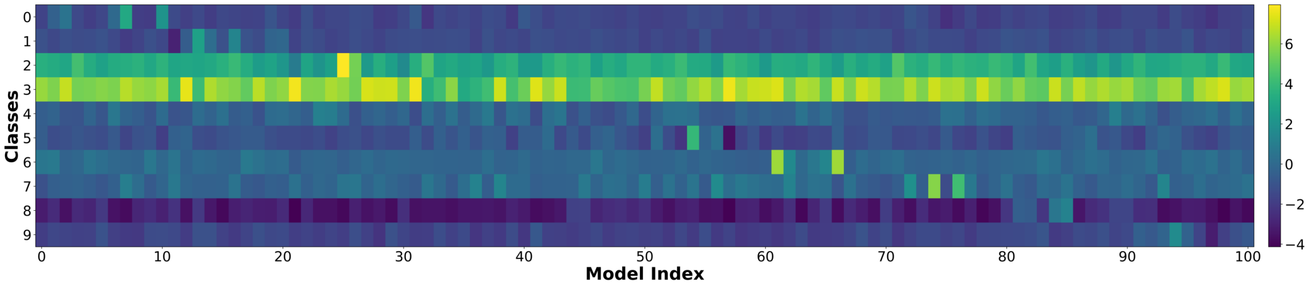


Figure 5. The results of DF-TND in detecting high-risk models (8-bit quantized ResNet-18 models on CIFAR-10). In the heatmap, each row corresponds to a class, and each column represents a target model. The colors in the heatmap indicate the values of the logit increases. The leftmost one (with model index '0') is the result of detecting the M_o model. From 1 to 100, every successive 10 models are obtained by targeting images belonging to the same class.

Table 5. The resistance to fine-tuning on CIFAR-10. The ASR in the quantization column denotes the ratio of cases that flipping the fine-tuned model according to $M_f - M_r$ can still succeed.

Method	Quantization	ACC (%)	ASR (%)	N_{flip}
Fine-tuning	ResNet-18	92.46±2.50	99.7	9.88±14.97
FSA	8-bit	91.08±3.10	100	6.15±8.42
T-BFA		90.70±3.05	100	5.11±6.50
TA-LBF	ASR: 31.8%	90.74±3.52	100	3.49±2.52
Fine-tuning	VGG-16	88.80±3.76	81.3	3.27±5.01
FSA	8-bit	88.33±8.09	100	3.60±5.17
T-BFA		87.50±4.05	100	1.32±0.82
TA-LBF	ASR: 51.3%	88.57±3.76	100	3.15±2.45
Fine-tuning	ResNet-18	89.97±2.41	99.5	17.39±24.39
FSA	4-bit	88.38±3.16	100	7.97±9.82
T-BFA		88.21±2.91	100	5.72±6.24
TA-LBF	ASR: 21.2%	88.47±3.36	100	3.77±2.24
Fine-tuning	VGG-16	86.22±4.32	84.7	7.86±8.51
FSA	4-bit	75.80±8.13	99.6	8.41±8.41
T-BFA		83.70±4.1	100	2.36±1.53
TA-LBF	ASR: 25.7%	85.73±3.86	100	4.26±2.00

DF-TND identified suspicious models according to the logit increase. The results of CIFAR-10 in Figure 5 show that logit increases are all below its suggested threshold ($i.e.$, 100), indicating that *all released high-risk models are regarded as benign*. Besides, the patterns of logit increase of the 100 M_r models are similar to that of the original model M_o (with index '0'). It is mostly because high-risk models are usually obtained by flipping limited bits of M_o, resulting in minor differences between M_o and M_r in their performance. In conclusion, *our TBA is resistant to DF-TND*. These results verify the stealthiness of our method.

The Resistance to Fine-tuning. Except for model-level detection, the victim users may adopt their local benign samples to fine-tune the released model before deployment. This method may be effective in defending against our attack since it can change the decision surface. We adopt 128 benign samples to fine-tune each released model 5,000 iterations with the learning rate set as 0.1. As shown in Table 5, fine-tuning can indeed reduce our attack success rate from nearly 100% to 30% on average. However, for those failed cases where we cannot trigger malicious behavior via flipping the differences between M_f and M_r of tuned models, the adversaries can still adopt existing bit-flip methods via flipping significantly fewer critical bits (compared to the case of attacking the original model) for the attack (as shown in the last column of Table 5). As such, our TBA is also resistant to fine-tuning to some extent.

5. Conclusion

In this paper, we revealed the potential limitation of existing bit-flip attacks (BFAs) that they still need a relatively large number of bit-flips to succeed. We argued that it is mostly because the victim model may be far away from its malicious counterparts in the parameter space. Motivated by this understanding, we proposed the training-assisted bit-flip attack (TBA) as a new and complementary BFA paradigm where the adversary is involved in the training stage to build a high-risk model to release. We formulated this problem as an instance of multi-task learning, where we jointly optimized a released model and a flipped model with the minimum distance so that the former one is benign and the latter is malicious. We also proposed an effective method to solve this problem. We hope this paper can provide a deeper insight into bit-flip attacks, to facilitate the design of more effective defenses and secure DNNs.

Although we have not yet found an effective defense in this paper, one can at least alleviate or even avoid this threat from the source by using trusted models solely and monitoring the deployment stage. Our next step is to design principled and advanced defenses against TBA.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB3105200) and the National Natural Science Foundation of China (62106127). This work was partly done when Yiming Li was a research intern at Ant Group and a Ph.D. student at Tsinghua University while he is currently a research fellow at Zhejiang University.

References

[1] Jiawang Bai, Bin Chen, Yiming Li, Dongxian Wu, Weiwei Guo, Shu-tao Xia, and En-hui Yang. Targeted attack for deep hashing based retrieval. In *ECCV*, 2020.

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

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

[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. *ICLR*, 2021.

[5] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

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

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

[8] Bangyan He, Jian Liu, Yiming Li, Siyuan Liang, Jingzhi Li, Xiaojun Jia, and Xiaochun Cao. Generating transferable 3d adversarial point cloud via random perturbation factorization. In *AAAI*, 2023.

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

[10] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

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

[12] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

[14] Zhifeng Li, Dihong Gong, Yu Qiao, and Dacheng Tao. Common feature discriminant analysis for matching infrared face images to optical face images. *IEEE transactions on image processing*, 23(6):2436–2445, 2014.

[15] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *ICML*, 2016.

[16] Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, 2017.

[17] Xiangyu Qi, Tinghao Xie, Yiming Li, Saeed Mahloujifar, and Prateek Mittal. Revisiting the assumption of latent separability for backdoor defenses. In *ICLR*, 2023.

[18] Haibo Qiu, Baosheng Yu, Dihong Gong, Zhifeng Li, Wei Liu, and Dacheng Tao. Synface: Face recognition with synthetic data. In *ICCV*, 2021.

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

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

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

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

[23] 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.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[25] Mohammad Samragh, Mojan Javaheripi, and Farinaz Koushanfar. Codex: Bit-flexible encoding for streaming-based fpga acceleration of dnns. *arXiv preprint arXiv:1901.05582*, 2019.

[26] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*, 2018.

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

[28] Gowthami Somepalli, Liam Fowl, Arpit Bansal, Ping Yeh-Chiang, Yehuda Dar, Richard Baraniuk, Micah Goldblum, and Tom Goldstein. Can neural nets learn the same model twice? investigating reproducibility and double descent from the decision boundary perspective. In *CVPR*, 2022.

[29] Disong Wang, Songxiang Liu, Xixin Wu, Hui Lu, Lifa Sun, Xunying Liu, and Helen Meng. Speaker identity preservation in dysarthric speech reconstruction by adversarial speaker adaptation. In *ICASSP*, 2022.

[30] 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.

[31] Yitong Wang, Dihong Gong, Zheng Zhou, Xing Ji, Hao Wang, Zhifeng Li, Wei Liu, and Tong Zhang. Orthogonal deep features decomposition for age-invariant face recognition. In *ECCV*, 2018.

[32] Baoyuan Wu and Bernard Ghanem. ℓ_p-box admm: A versatile framework for integer programming. *IEEE transactions on pattern analysis and machine intelligence*, 41(7):1695–1708, 2018.

[33] Haibin Wu, Lingwei Meng, Jiawen Kang, Jinchao Li, Xu Li, Xixin Wu, Hung-yi Lee, and Helen Meng. Spoofing-aware speaker verification by multi-level fusion. In *Interspeech*, 2022.

[34] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

[35] 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.

[36] Tongqing Zhai, Yiming Li, Ziqi Zhang, Baoyuan Wu, Yong Jiang, and Shu-Tao Xia. Backdoor attack against speaker verification. In *ICASSP*, 2021.

[37] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *DAC*, 2019.

Supplementary Materials:
One-bit Flip is All You Need: When Bit-flip Attack Meets Model Training

1. Algorithm Outlines

Algorithm 1 An effective solution to the BIP

```
Input: The original quantized DNN model $f$ with weights $\mathbf{\Theta}$, $\mathbf{B}_o$, target sample $\boldsymbol{x}^*$ with source label $s$, target class $t$, auxiliary sample set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, hyper-parameters $\lambda_1$, $\lambda_2$ and $k$.
Output: $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$.
Initialize $\hat{\boldsymbol{b}}^0$, $\boldsymbol{b}^0$, $\boldsymbol{u}_1^0$, $\boldsymbol{u}_2^0$, $\boldsymbol{u}_3^0$, $\boldsymbol{u}_4^0$, $\boldsymbol{z}_1^0$, $\boldsymbol{z}_2^0$, $\boldsymbol{z}_3^0$, $\boldsymbol{z}_4^0$;
Let $r \leftarrow 0$ ;
while not converged do
    Update $\hat{\boldsymbol{b}}^{r+1}$;
    Update $\boldsymbol{u}_1^{r+1}$ and $\boldsymbol{u}_2^{r+1}$;
    Update $\boldsymbol{b}^{r+1}$;
    Update $\boldsymbol{u}_3^{r+1}$ and $\boldsymbol{u}_4^{r+1}$;
    Update $\boldsymbol{z}_1^{r+1}$, $\boldsymbol{z}_2^{r+1}$, $\boldsymbol{z}_3^{r+1}$ and $\boldsymbol{z}_4^{r+1}$;
    $r \leftarrow r + 1$.
end while
```

2. Experiment Setups

Target Models. We provide information about target models which are in the floating-point form before quantization.

Table 1. Information of target models.

Dataset	Model	Accuracy (%)	Number of all parameters	Number of target parameters
CIFAR-10	ResNet-18	95.25	11,173,962	1,024
	VGG-16	93.64	14,728,266	1,024
ImageNet	ResNet-34	73.31	21,797,672	1,024
	VGG-19	74.22	143,678,248	8,192

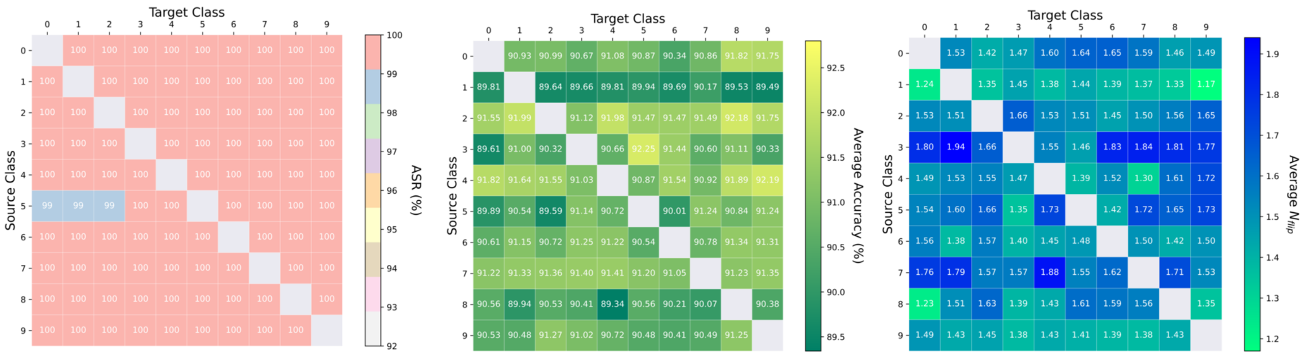
Detailed Settings of TBA. Having described how the hyperparameters λ_1, λ_2, k, and N are set, we provide the detailed configuration of the hyperparameters associated with the ℓ_p-Box ADMM algorithm. To begin, we duplicate the parameters of the last fully-connected layer twice to obtain the target parameters $\hat{\boldsymbol{b}}^0$ and $\boldsymbol{b}^0$. We then initialize the additional parameters and the dual parameters by assigning $\boldsymbol{u}_1^0$, $\boldsymbol{u}_2^0$, $\boldsymbol{u}_3^0$, $\boldsymbol{u}_4^0$ to $\boldsymbol{b}^0$ and setting $\boldsymbol{z}_1^0$, $\boldsymbol{z}_2^0$, $\boldsymbol{z}_3^0$, $\boldsymbol{z}_4^0$ to $\mathbf{0}$. During the process, we adopt a learning rate of 0.005 and 0.01 in CIFAR-10 and ImageNet, respectively, to update $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$ for three inner rounds in each iteration. The optimization process is allowed to continue for up to 2,000 iterations. For the ADMM algorithm, the penalty parameters ρ_1, ρ_2, ρ_3 and ρ_4 are identically set to 0.0001 and increase by multiplying a factor of 1.01 every iteration until a maximal value of 50 is reached. From all candidates couples of $\hat{\boldsymbol{b}}^i$ and $\boldsymbol{b}^i$, we select the closest couple that can classify the target sample $\boldsymbol{x}^*$ to the target class t and the source class s, respectively. Note that no additional samples are used to appropriate the accuracy of candidate models when choosing M_r and M_f. The optimization process will end if one of the following three conditions is met:

- The maximal number of 2,000 iterations is reached.
- No improvement is gained for 300 iterations.
- The constraints $\hat{\boldsymbol{b}} = \boldsymbol{u}_1$, $\hat{\boldsymbol{b}} = \boldsymbol{u}_2$, $\boldsymbol{b} = \boldsymbol{u}_3$ and $\boldsymbol{b} = \boldsymbol{u}_4$ are all satisfied with distance less than 0.0001.

Implementation Details of Baselines. We include four baseline attacks to compare with our TBA. We try our best to make the experiment settings fair for all attacks. Besides fixing target models and target samples the same, we provide the same 128 and 512 auxiliary samples respectively in CIFAR-10 and ImageNet for each attack. To align with our threat model, we adjust their attack goals to the same sample-wise targeted attack as our TBA. Fine-tuning and FSA [3] are all designed for updating the parameters of full-precision floating-point models. Since the target model has been deployed, its step size should be fixed, causing the invalidity of quantization-aware training. We adjust these two methods to directly attack models which have been quantized to 4/8 bit-width. Fixing the step size of the target model unchanged, we optimize the parameter in the grain of each bit continuously while testing the attack performance and calculate the N_{flip} discretely by transforming the bits to 0-1 form in the following way:

$$b = \begin{cases} 1 & \text{if } x \geq \frac{1}{2}, \\ 0 & \text{if } x < \frac{1}{2}. \end{cases} \tag{1}$$

We adopt the l_0-regularized form of FSA [3], which can help limit the increment of N_{flip} in theory. T-BFA [2] is a class-specific targeted attack, which aims to misclassify samples from the source class as the target class. We transform it into a sample-specific attack and restrict it only to attacking the bits of the final fully-connected layer. TA-LBF, which also involves an ADMM-based optimization process, gets all hyperparameters strictly following [1]. In the sce-



(a) ASR (b) ACC (c) N_{flip}

Figure 1. Results of sensitivity to different source and target classes. In these heatmaps, each row stands for a source label and each column represents a target label. The value in cell (i, j) is calculated by averaging those of 100 attack instances with source class as i and target class as j. The lighter color means a better result.

nario of deployment-stage attacks, the original model M_o is released. ASR is the ratio of the cases where a malicious model can be successfully obtained utilizing baseline attacks, ACC is the averaged accuracy of all post-attack malicious models, and N_{flip} is the averaged number of bit-flips that is required to convert M_o to the malicious model.

3. Exploratory Experiments

3.1. Sensitivity to Different Target Classes

In the main experiments, we randomly assign target class t for each selected target sample $\boldsymbol{x}^*$, the good results of which demonstrate that the performance of TBA is not dependent on the choice of the target sample and target class. In this part, we further explore the impact of target class t on the performance of TBA at the label level. To achieve it, we choose 100 random samples from each class of the CIFAR-10 dataset, and utilize TBA to misclassify them to the other nine classes. The final results are shown in Figure 1. With the default settings, TBA can attain an almost 100% attack success rate regardless of the choice of target class. The choice of target class influences the ACC of attacked model M_f a lot. For example, observing the fourth row of Figure 1(b), we find that the ACC drops sharply when misclassifying samples collected from class 3 to class 0 compared to other choices of target class. Besides, the performance of TBA is concerned with the choice of source class as well. Attacking samples of class 2 can always render models with high accuracy while attacking those of class 3 will yield models with relatively low accuracy. N_{flip}, which is 1.17 in best cases and 1.94 in worst cases, is also related to the choice of source and target class. The differences in ACC and N_{flip} can be attributed to the risk level of the target model. We assume that target model is naturally at high risk when faced with certain target samples, due to its imbalanced ability to predict samples of different classes. In conclusion, the performance of TBA is related to but not dependent on the choice of target class.

3.2. Loss Curve of the Optimization Process

As stated in Section 3.3 of the main manuscript, $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$ get alternately updated in each iteration. So we observe the loss curve respectively after $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$ get updated in the i-th iteration. As shown in Figure 2, at the start of the optimization process, it is inevitable that the accuracy-related loss term $\mathcal{L}_b$ increases a little since $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$ are moving towards a high-risk area. At the rest of the process, $\mathcal{L}_b$ remains at an acceptable level with the help of auxiliary set $\mathcal{D}$. The loss term $\mathcal{L}_d$, which measures the distance between $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$, keeps fairly small during most of the optimization process, which demonstrates that $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$ are closely bond across the process and satisfies the requirements for efficiency as wanted. The loss term $\mathcal{L}_m$ and the loss term $\mathcal{L}_i$, which respectively force the $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$ to classify the target sample $\boldsymbol{x}^*$ to target class t and ground-truth class s show reverse patterns in the two curves because these two terms are just optimized respectively by updating $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$. Taking $\mathcal{L}_m$ as an example, it is minimized when updating $\hat{\boldsymbol{b}}$. However, when $\boldsymbol{b}$ gets updated, $\hat{\boldsymbol{b}}$ will be attracted to follow it for the existence of the distance-related loss term $\mathcal{L}_d$, in which case, $\mathcal{L}_m$ will probably become larger. In conclusion, the updates of $\hat{\boldsymbol{b}}$ and $\boldsymbol{b}$ will take over the optimization process in turn, causing its related loss terms minimized but its unrelated loss terms to fluctuate. In several cases, the $\mathcal{L}_i$ ends up with a high value for that $\boldsymbol{b}$ can be conducted by $\hat{\boldsymbol{b}}$ to the side of malicious parameters.

3.3. Statistics of Running Time

We analyze the running time of the three standard bit flip attacks against quantized models, whose official codes can be accessed. We present the average time used to attack an 8-bit quantized ResNet-18 model with 1,000 different target samples. As shown in Table 2, in CIFAR-10, the heuristic method T-BFA outperforms the two optimization-based

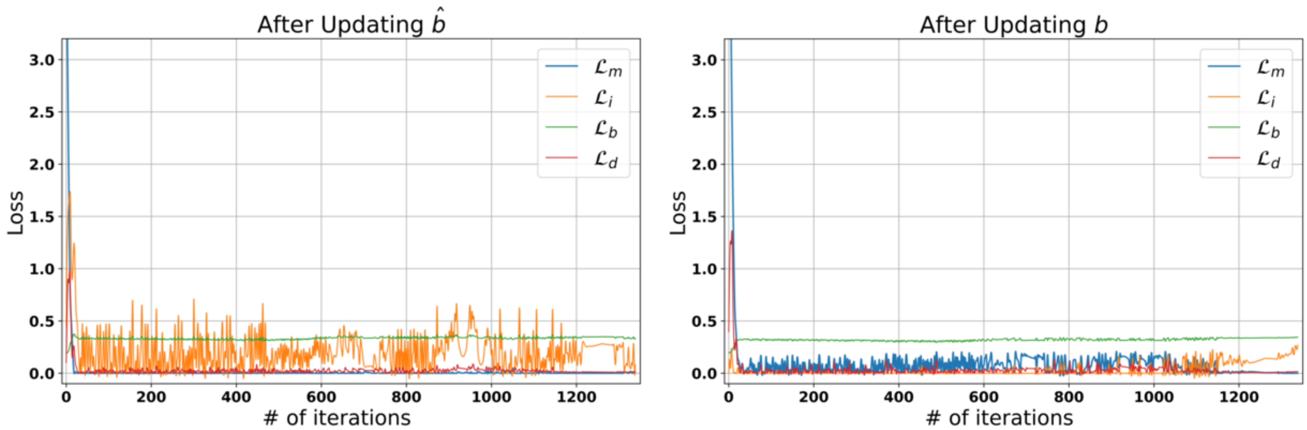


Figure 2. Loss curves.

Table 2. The running time of attacks.

Dataset	Model	Time Cost (s)		
		T-BFA	TA-LBF	TBA (ours)
CIFAR-10	ResNet-18	12.68	673.49	16.07
	VGG-16	3.43	214.65	15.11
ImageNet	ResNet-34	30.88	205.71	25.12
	VGG-19	159.86	258.18	76.79

methods, TA-LBF and TBA. The running time of T-BFA is highly correlated with the number of bit-flips for it will flip bits one by one until success. For example, attacking VGG-16 utilizing T-BFA costs only 3.43 seconds because it needs only 8.75 bit-flips on average to succeed. For the two optimization-based methods, the time to finish a complete optimization process of TBA is approximately twice that of TA-LBF because the number of parameters involved in TBA is twice that in TA-LBF. However, TA-LBF has to determine suitable hyperparameters in the manner of grid search, making it more time-consuming. For ImageNet, it is usually required to flip more bits to succeed, and our TBA performs better than the other two methods in time efficiency, which can further demonstrate its threat in more complicated tasks. Note that attacking ResNet-34 is more costly than attacking VGG-19 because VGG-19 has a larger number of target parameters as shown in Table 1.

3.4. Training-assisted Baselines

In the main experiments, we compared only to deployment-only BFAs since training-assisted extension is one of our core contributions. However, we also consider comparing our TBA to the training-assisted variants of T-BFA and TA-LBF (FT and FSA cannot be extended) on CIFAR-10 with 8-bit quantized VGG. As shown in Table 3, TBA is on par with or even better than all training-assisted baselines on all metrics.

Table 3. Comparison to the training-assisted variants of baselines. Data points marked in red denote a relatively worse performance.

Method	ASR (%)	N_{flip}-r	ACC (M_r)	N_{flip}-f	ACC (M_f)	Time (s)
T-BFA	100	7.75	90.73	1.01	87.84	38.14
TA-LBF	78.3	7.67	92.66	1.15	89.23	59.89
TA-LBF-GS	97	10.33	92.93	1.01	90.53	545.26
Ours	100	11.25	92.43	1.04	89.03	39.02

4. Discussions About the Threat Model

Our approach differs from the previous BFAs in that we assume the adversary has the access to the training stage and further has the ability to decide the model to be released, which provides a valid reason for the white-box setting generally postulated but without detailed explanation in deployment-time bit flip attacks. In prior BFAs, third-party adversaries usually utilize white-box information like gradients to search for critical bits of the target model's parameters to inject malicious functionality.

We assume the adversary can implement such a training-assisted attack in at least two cases: (1) The adversary is an insider of one development project, who is in nature able to manipulate the training stage; (2) Utilizing outsourced models is a common phenomenon in the domain of deep learning. In this case, similar to the scenario of backdoor attacks, the adversary can act as an outsider, who releases a high-risk model M_r to the Internet and waits for the victim users to download and then deploy it.

References

[1] 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. *ICLR*, 2021.

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

[3] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *DAC*, 2019.