Abstract—Many attack paradigms against deep neural networks have been well studied, such as the backdoor attack in the training stage and the adversarial attack in the inference stage. In this article, we study a novel attack paradigm, the bit-flip based weight attack, which directly modifies weight bits of the attacked model in the deployment stage. To meet various attack scenarios, we propose a general formulation including terms to achieve effectiveness and stealthiness goals and a constraint on the number of bit-flips. Furthermore, benefiting from this extensible and flexible formulation, we present two cases with different malicious purposes, i.e., single sample attack (SSA) and triggered samples attack (TSA). SSA which aims at misclassifying a specific sample into a target class is a binary optimization with determining the state of the binary bits (0 or 1); TSA which is to misclassify the samples embedded with a specific trigger is a mixed integer programming (MIP) with flipped bits and a learnable trigger. Utilizing the latest technique in integer programming, we equivalently reformulate them as continuous optimization problems, whose approximate solutions can be effectively and efficiently obtained by the alternating direction method of multipliers (ADMM) method. Extensive experiments demonstrate the superiority of our methods.

Index Terms—Bit-flip, weight attack, deep neural networks, vulnerability, binary optimization, mixed integer programming.

I. INTRODUCTION

Deep neural networks (DNNs) have achieved state-of-the-art performance in computer vision [1], natural language processing [2], robotic manipulation [3], etc. However, many works [4], [5] have revealed that DNNs are vulnerable to a range of attacks, which has attracted great attention, especially for security-critical applications (e.g., face recognition [6], [7], [8], medical diagnosis [9], [10], and autonomous driving [11], [12]). For example, backdoor attack [13], [14] manipulates the behavior of the DNN model by mainly poisoning some training data in the training stage; adversarial attack [15], [16] aims to fool the DNN model by adding malicious imperceptible perturbations onto the inputs in the inference stage.

Besides the above backdoor attack and adversarial attack, there is a novel attack paradigm, i.e., \textit{weight attack} [17], posing a new security threat to deployed DNNs. It assumes that the attacker can directly change the parameters of a deployed model to achieve some malicious purposes. Because the deployed DNN model is stored as binary bits in the memory, the attacker can flip the weight bits using some physical fault injection techniques, such as Row Hammer Attack [18], [19] and Laser Beam Attack [20], making the weight attack realistic. Since it neither modifies the training data nor controls the training process and performs in the deployment stage, it is difficult for both the service provider and the user to realize the existence of the attack. The comparison of backdoor attack, adversarial attack, and weight attack is presented in Table I.

In general, for the weight attack, the attacker’s goals include two aspects: \textit{effectiveness} and \textit{stealthiness}. The effectiveness requires that the attacked DNN can meet the attacker-specified malicious purpose (which will be detailed later), while the stealthiness encourages that the attacked DNN behaves normally on samples except the attacked one(s). Moreover, since physical bit flipping techniques can be time-consuming [19], [21] and more bit-flips are easier to be detected [22], an efficient and practical bit-flip based weight attack requires flipping less number of bits.

Some previous works [23], [24], [25] have explored the bit-flip based weight attack with different malicious purposes, such as misclassifying samples with a trigger to a target class [24] and misclassifying clean samples from a source class to a target class [25]. Their efforts partially focused on how to identify the critical bits in a large number of model parameters. These
methods mostly use some heuristic strategies. For example, the attack in [24] uses the gradient ranking to select the candidate flipped bits, which are important to misclassify samples with a trigger. However, there is a large margin between the performance of solutions obtained by these heuristic strategies and that of the optimal solutions, e.g., more than 600 bits needed in [24]. Besides, it is difficult to design a strategy that can be applied to various malicious purposes, and thus these works [23], [24], [25] for different purposes need to design strategies separately. Therefore, it is desirable to propose a versatile method, which can improve existing attacks and handle different malicious purposes.

In this work, to address the above problem, we propose a versatile weight attack based on a general formulation. We formulate effectiveness and stealthiness goals as two terms in the objective function, respectively. Specifically, the stealthiness loss encourages the correctness of classifying samples except the attacked one(s), and the effectiveness loss can be customized depending on the attacker’s malicious purpose, which will be illustrated in our two special cases below. In addition, to flip fewer bits, we leverage a cardinality constraint which requires that the number of bit-flips is less than a preset value. To adapt to different attack scenarios, our formulation contains two groups of optimized variables: weight bits in the memory as binary variables and learnable parameters of the sample modification. Moreover, we limit the bit-flips in the last fully-connected layer since it directly influences the output, which results in better performance as discussed in Section IV-E6.

We introduce two effectiveness goals to apply our general formulation to two attack scenarios, resulting in two special cases: single sample attack (SSA) and triggered samples attack (TSA). Specifically, SSA aims at misclassifying a specific sample to a predefined target class without any sample modification but relying on the bit flipping, and thus it is a binary optimization. TSA is to misclassify the samples embedded with a trigger [26], which is a mixed integer programming (MIP) due to the flipped bits and the learnable trigger. Their goals are demonstrated in Fig. 1.

However, how to optimize the binary variables with a cardinality constraint is a challenging problem. Fortunately, inspired by an advanced optimization method, the \(\ell_p\)-box ADMM [27], we can reformulate SSA and TSA as continuous optimization problems, whose approximate solutions can be efficiently and effectively obtained by the alternating direction method of multipliers (ADMM) [28], [29]. Consequently, the flipped bits can be determined through optimization rather than the heuristic strategy, making our attack more effective. We conduct experiments on the quantized DNN models following the settings in recent works [23], [24], and we also extend our method to attack full-precision networks. Extensive experiments demonstrate the superiority of the proposed method over several existing weight attacks.\(^1\) For example, in attacking an 8-bit quantized ResNet-18 on ImageNet, on average, SSA achieves a 100% attack success rate with 7.37 b-flips and 0.09% accuracy degradation of the rest unspecified inputs, and TSA achieves a 95.63% attack success rate with 3.4 b-flips and 0.05% accuracy degradation of the original samples. Moreover, we demonstrate that the proposed method is more resistant to existing defense methods.

This work builds upon the preliminary conference paper [30], which primarily focuses on attacking a specified sample without modifying the attacked sample. The new major contributions are summarized as follows.

1) We expand the attack in the conference paper to a general formulation, which can be flexibly applied to different attack scenarios. This general formulation consists of identifying the flipped bits and learning the modification on the samples.
2) We propose two special cases of our general formulation, where SSA is the attack in our conference paper and TSA is the new one. Unlike SSA, TSA aims at misclassifying all samples with the learned trigger.
3) Our method is extended to attack full-precision DNNs for the sake of being more versatile, while the conference version only considers the quantized DNNs.
4) Utilizing the proposed optimization method, we conduct extensive experiments to evaluate our attacks. In particular, the new attack, TSA, reduces the number of bit-flips greatly compared to previous methods.

**II. RELATED WORK**

We discuss related works about the weight attack here, and include ones about the adversarial attack and the backdoor attack in Appendix A, available online, which are also related to this work.

Weight attack [31], [32], [33], [34] modifies model parameters in the deployment stage. It received extensive attention, since the

\(^1\)Our code is available at: https://github.com/jiawangbai/Versatile-Weight-Attack.
robust model weight is the cornerstone of employing DNNs in many security-critical applications [35]. First, two schemes are proposed in [31] to modify model parameters for misclassification without and with considering stealthiness, which is dubbed single bias attack (SBA) and gradient descent attack (GDA) respectively. After that, Trojan attack [32] and model-reuse attack [36] were proposed, which inject malicious behavior to the DNN by retraining the model. These methods require to change lots of parameters. Recently, fault sneaking attack (FSA) [21] was proposed, which aims to misclassify certain samples into a target class by modifying the parameters with two constraints, including maintaining the classification accuracy of other samples and minimizing parameter modifications.

**Bit-Flip Based Attack:** Recently, some physical fault injection techniques [18], [19], [20] were proposed, which can be adopted to precisely flip any bit in the memory. Those techniques promote researchers to study how to modify model parameters at the bit-level. As a branch of weight attack, the bit-flip based attack was first explored in [23]. It proposed an untargeted attack that can convert the attacked DNN to a random output generator with several bit-flips. Besides, the targeted bit Trojan (TBT) [24] injects the fault into DNNs by flipping some critical bits. Specifically, the attacker flips the identified bits to force the network to classify all samples embedded with a trigger to a certain target class, while the network operates with normal inference accuracy with benign samples. Most recently, the targeted bit-flip attack (T-BFA) [25] achieves malicious purposes without modifying samples. It is worth noting that the above bit-flip based attacks leverage heuristic strategies to identify critical weight bits. How to find critical bits for the bit-flip based attack method is still an important open question.

To mitigate the affect of the weight attack, many works have investigated the defense mechanisms. Previous studies [25], [37] observed that increasing the network capacity can improve the robustness against the bit-flip based attack. Moreover, the training strategies in [37], [38], [39] can improve the robustness of the model parameters. There exist other works considering defense in the inference stage, such as weight reconstruction-based defense [40], detection-based defense [41], [42], and Error Correction Codes (ECC)-based defense [43], [44]. For a more comprehensive comparison, we also evaluate the resistance of attack methods to defense strategies in [25], [37].

### III. METHODOLOGY

#### A. Preliminaries

**Storage and Calculation of Quantized DNNs:** Currently, it is a widely-used technique to quantize DNNs before deploying on devices for efficiency and reducing storage size. For each weight in the layer of a $Q$-bit quantized DNN, it will be represented and then stored as the signed integer in two’s complement representation ($v = [v_Q; v_{Q-1}; \ldots; v_1] \in \{0, 1\}^Q$) in the memory. The attacker can modify the weights of DNNs by flipping the stored binary bits. In this work, we adopt the layer-wise uniform weight quantization scheme similar to Tensor-RT [45]. Accordingly, each binary vector $v$ can be converted to a real number by a function $h(\cdot)$

$$h(v) = \left(-2^{Q-1} \cdot v_Q + \sum_{i=1}^{Q-1} 2^{i-1} \cdot v_i\right) \cdot \Delta^l,$$

(1)

where $l$ indicates which layer the weight is from, $\Delta^l > 0$ is a known and stored constant which represents the step size of the $l$th layer weight quantizer.

**Notations:** We denote a $Q$-bit quantized DNN-based classification model as $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ being the input space and $\mathcal{Y} \in \{1, 2, \ldots, K\}$ being the $K$-class output space. Assuming that the last layer of this DNN model is a fully-connected layer with $\mathbf{B} \in \{0, 1\}^{K \times C \times Q}$ being the quantized weights, where $C$ is the dimension of last layer’s input. Let $\mathbf{B}_{i,j} \in \{0, 1\}^Q$ be the two’s complement representation of a single weight and $\mathbf{B}_i \in \{0, 1\}^{C \times Q}$ denotes all the binary weights connected to the $i$th output neuron. Given a test sample $x_i$ with the ground-truth label $y_i$, $f(x_i; \Theta, \mathbf{B}) \in [0, 1]^K$ is the output probability vector and $g(x_i; \Theta) \in \mathbb{R}^C$ is the input of the last layer, where $\Theta$ denotes the model parameters except the last layer.

**Attacker’s Capacities:** In this article, we focus on the white-box bit-flip based weight attack, which was first introduced in [23]. Specifically, we assume that the attacker has full knowledge of the edge of the model (including its architecture, parameters, and parameters’ location in the memory), and can precisely flip any bit in the memory. Besides, we assume that attackers can have access to a small portion of benign samples, but they cannot tamper with the training process and the training data.

**Attack Scenarios:** We consider two specific scenarios in Sections III-C and III-D. The scenario in Section III-C requires that the attacked model can misclassify a specific sample to a predefined target class without any sample modification and the prediction accuracy of other samples will not be significantly reduced. This scenario is realistic for the attacker who wants to manipulate the behavior of the DNNs on specific inputs, which was also considered in [21], [25], [31]. In Section III-D, we consider the attack scenario in [24] where the attacked model can classify the samples embedded with a designed trigger (e.g., a square patch) [26], [46] to a target class, and performs accurate classification on most inputs when the trigger is removed. Misclassifying any samples with the trigger may be suitable for the attacker who requires various malicious inputs. For example, in the context of autonomous driving, the attacker wishes that autonomous cars recognize all road signs with the trigger as the stop sign.

Note that in both scenarios, following [21], [24], [25], [31], we consider the stealthiness which requires that the attacked model should classify samples except the attacked one(s) correctly. Besides, the stealthiness is discussed from another perspective in Section IV-E4, i.e., unchanging the predictions of the original model.

#### B. General Formulation

In this section, we give the general formulation of the proposed weight attack, which will be specified as two cases to
handle the scenarios in Sections III-C and III-D. As mentioned in Section III-A, we assume that the attacker can get access to two sample sets to perform attack: \( D_1 = \{ (x_i, y_i) \}_{i=1}^{N_1} \) contributes to achieve the attack effectiveness and \( D_2 = \{ (x_i, y_i) \}_{i=1}^{N_2} \) helps to keep the attack stealthiness. Let the vectorized binary parameters \( \hat{b} \in \{0, 1\}^V \) being a part of the attacked model (e.g., the parameters of the last layer \( B \)) which the attacker intends to modify, and \( \hat{b} \in \{0, 1\}^V \) corresponds to the modified version of \( b \). The dimension \( V \) will be specified later. When performing attack, we only modify \( b \) and the remaining parameters of the attacked model are fixed. Note that we will omit the fixed parameters of the attacked model for clarity in the below formulas. Let \( \phi \) with the learnable parameter \( q \) represent the modification operator on \( D_1 \), which is supposed to be differentiable w.r.t. \( q \). The overall objective function is below

\[
\min_{b, q} \lambda_1 L_1(\phi(D_1; q), t; \hat{b}) + \lambda_2 L_2(D_2; \hat{b}),
\]

s.t. \( \hat{b} \in \{0, 1\}^V \), \( d_H(\hat{b}, b) \leq k \),

where \( d_H(\cdot, \cdot) \) denotes the Hamming distance, and \( \lambda_1, \lambda_2 > 0 \) are the trade-off parameters. The loss \( L_1 \) is used to ensure the attack effectiveness, which can be customized according to the attacker’s purpose. By minimizing \( L_1 \), the attacked model can misclassify the attacked sample(s) (e.g., a specified sample or samples with a trigger) into a target class \( t \), as shown in Fig. 1. We will detail the loss \( L_1 \) later of two special cases, respectively.

The attack may be easily detectable only using \( L_1 \), since the attacked model behaves abnormally even on samples except the attacked one(s). Therefore, we utilize the loss \( L_2 \) to ensure the attack stealthiness, as follows,

\[
L_2(D_2; \hat{b}) = \sum_{(x_i, y_i) \in D_2} \ell(f(x_i; \hat{b}), y_i),
\]

where \( f_j(x_i; \hat{b}) \) indicates the posterior probability of \( x_i \) w.r.t. class \( j \), \( \ell(\cdot, \cdot) \) is specified by the cross entropy loss.

The cardinality constraint \( d_H(\hat{b}, b) \leq k \) to reduce the number of bit-flips, where \( k > 0 \) is a hyper-parameter. We limit the number of bit-flips for two reasons. One is that physical bit flipping techniques can be time-consuming as discussed in [19], [21]. For instance, on a machine, Sandy Bridge i3-2120, using the row hammer attack in [47] can flip 4 or 5 bits per hour. Besides, to locate bits, some techniques try to test every position to find a desired one [22], [48], which takes a long time. Another reason is that bit flipping leads to abnormal behaviors in the attacked system. Some techniques generate conspicuous memory footprints [19], [49] or suspicious cache activity of processes [47], [48]. As a result, they may be detected by some physical detection-based defenses, and more bit-flips improve the probability of being detected [19], [50].

C. Special Case I: Single Sample Attack

In this section, based on the above general formulation, we introduce our first type of attack: single sample attack (SSA).

**Loss for Ensuring Effectiveness:** Recall that SSA aims at forcing a specific image to be classified as the target class by modifying the model parameters at the bit-level. To this end, the most straightforward way is maximizing the logit of the target class while minimizing that of the source class. For a sample \( x \) with the ground-truth label \( s \), the logit of a class can be directly determined by the input of the last layer \( g(x; \Theta) \) and weights connected to the node of that class. Accordingly, we can modify weights only connected to the source and target class to fulfill our purpose. Moreover, for SSA, we specify the set \( D_1 \) as \( \{ (x, s) \} \) and use \( x \) and \( s \) straightly in the below formulations for clarity. We have no modification on the attacked sample, i.e., \( \phi \) in (2) is specified as \( \phi(\{ (x, s) \}, q) = \{ (x, s) \} \) and will be omitted below. The loss for ensuring effectiveness is as follows:

\[
L_1^{SSA}(x, s, t; \hat{B}_s, \hat{B}_t) = \max \left( \tau - p(x; \Theta, \hat{B}_s) + \delta, 0 \right) + \max \left( p(x; \Theta, \hat{B}_s) - \tau + \delta, 0 \right),
\]

where \( p(x; \Theta, \hat{B}_s) = \{h(\hat{B}_{1s}); h(\hat{B}_{2s}); \ldots; h(\hat{B}_{Ks})\}^\top g(x; \Theta) \) indicates the logit of class \( i \) \((i = s \text{ or } i = t)\), \( h(\cdot) \) is the function defined in (1), \( \tau = \max_{i \in \{0, \ldots, K\} \setminus \{s\}} p(x; \Theta, \hat{B}_i) \), and \( \delta \in \mathbb{R} \) indicates a slack variable, which will be specified in later experiments. The first term of \( L_1^{SSA} \) aims at increasing the logit of the target class, while the second term is to decrease the logit of the source class. The loss \( L_1^{SSA} \) is 0 only when the output on the target class is more than \( \tau + \delta \) and the output on the source class is less than \( \tau - \delta \). That is, the prediction on \( x \) of the target model is the target class \( t \).

**Loss for Ensuring Stealthiness:** \( L_2 \) defined by (3) can be rewritten for SSA using \( \hat{B}_s \) and \( \hat{B}_t \), as follows:

\[
L_2(D_2; \hat{B}_s, \hat{B}_t) = \sum_{(x_i, y_i) \in D_2} \ell(f(x_i; \hat{B}_s, \hat{B}_t), y_i).
\]

**Overall Objective for SSA:** In summary, the final objective function for SSA is as follows:

\[
\min_{\hat{B}_s, \hat{B}_t} \lambda_1 L_1^{SSA}(x, s, t; \hat{B}_s, \hat{B}_t) + \lambda_2 L_2(D_2; \hat{B}_s, \hat{B}_t),
\]

s.t. \( \hat{b} \in \{0, 1\}^{C \times Q} \), \( \hat{B}_s \in \{0, 1\}^{C \times Q} \), \( d_H(\hat{B}_s, \hat{B}_t) + d_H(B_t, \hat{B}_t) \leq k \).

The above objective function is one of the special forms of (2), where \( \phi \) is the identity function. \( \hat{B}_s, \hat{B}_t \in \{0, 1\}^{C \times Q} \) are two variables we want to optimize, corresponding to the weights of the fully-connected layer w.r.t. class \( s \) and \( t \), respectively, in the attacked DNN model. \( B_t \in \{0, 1\}^{K \times C \times Q} \) denotes the weights of the fully-connected layer of the original DNN model. Therefore, \( b \) in Section III-B corresponds to the reshaped and concatenated \( B_s \) and \( B_t \) and \( b \) could be the reshaped and concatenated \( B_s \) and \( B_t \), respectively. The size of \( b \) and \( \hat{b} \) is \( 2CQ \) (i.e., \( V = 2CQ \)).

D. Special Case II: Triggered Samples Attack

We present an attack which misclassifies the inputs with a learned trigger, namely, triggered samples attack (TSA).

**Loss for Ensuring Effectiveness:** For TSA, we can suppose that two sample sets \( D_1 \) and \( D_2 \) share the same data, i.e., \( D_1 = D_2 \) and \( N_1 = N_2 \). For simplicity, we use \( D = \)
\{(x_i, y_i)\}_{i=1}^{N} \text{ to denote these two sample sets in this section. To achieve the effectiveness goal, we present how to embed the inputs with a trigger first. Suppose that the trigger is a patch and its area is given by an attacker-specified mask } m \in [0, 1]^d.\text{ We define the function } \phi \text{ to generate the triggered samples, as follows.}

\[ \phi(D, q) = \{(1 - m) \odot x_i + m \odot q, y_i) \mid (x_i, y_i) \in D \}, \]

where \( \odot \) indicates the element-wise product and \( q \in \mathbb{R}^d \) is the trigger. Because the modification on the sample is differentiable w.r.t. \( q \), the trigger can be optimized using the gradient method to achieve a more powerful attack. Note that TSA aims at misclassifying the samples with the trigger from all classes, which is different from SSA. Therefore, we optimize the parameters of the last layer \( B \) and \( \bar{B} \) is the modified \( B \). We define the following objective to achieve the targeted misclassification of the triggered samples.

\[
\begin{align*}
L^TSA(\phi(D; q), t; \bar{B}) &= \sum_{(x_i, y_i) \in D} \ell(f((1 - m) \odot x_i + m \odot q; \bar{B}), t),
\end{align*}
\]

where \( \ell(\cdot) \) is the cross entropy loss. We can update the trigger \( q \) and \( \bar{B} \) alternatively to find the trigger and the bit-flips to minimize the above loss.

**Loss for Ensuring Stealthiness:** Since we modify \( B \) for TSA, \( L_2 \) defined by (3) can be rewritten as below.

\[
\begin{align*}
L_2(D; \bar{B}) &= \sum_{(x_i, y_i) \in D} \ell(f(x_i; \bar{B}), y_i).
\end{align*}
\]

**Overall Objective for TSA:** We summarize the objective function of TSA as below.

\[
\begin{align*}
\min_{B, q} \lambda_1 L^TSA(\phi(D; q), t; \bar{B}) + \lambda_2 L_2(D; \bar{B}) \\
\text{s.t. } \bar{B} \in \{0, 1\}^{K \times C \times Q}, \quad d_H(\bar{B}, \bar{B}) \leq k.
\end{align*}
\]

Note that we specify \( q \) as the learnable trigger pattern and the modification operator \( \phi \) as embedding this trigger (see (7)). Since the weights of the fully-connected layer \( \bar{B} \in \{0, 1\}^{K \times C \times Q} \) is the variable we want to optimize, \( \bar{B} \) in Section III-B corresponds to the reshaped \( \bar{B} \) and \( \bar{b} \) could be the reshaped \( B \). The size of \( b \) and \( \bar{B} \) is \( KCQ \) (i.e., \( V = KCQ \)).

**E. An Effective Optimization Method**

We present the optimization method for the challenging Problems (6) and (10), inspired by the generic solver for integer programming, \( \ell_p \)-Box ADMM [27]. For the sake of simplicity, we introduce our method using the general formulation (2) with learnable \( \bar{b} \) and \( q \).

The \( \ell_p \)-Box shows its superior performance in many tasks, e.g., model pruning [51], clustering [52], MAP inference [53], adversarial attack [54], etc. It replaces the binary constraint equivalently by the intersection of two continuous constraints, as follows

\[
\bar{b} \in [0, 1]^V \iff \bar{b} \in (S_b \cap \bar{S}_b),
\]

where \( \bar{S}_b = [0, 1]^V \) indicates the box constraint, and \( \bar{S}_p = \{\bar{b} : \|\bar{b} - \frac{1}{2}\|^2 = \frac{1}{4}\} \) denotes the \( \ell_2 \)-sphere constraint. Utilizing (11), problems with binary variables can be equivalently reformulated. Besides, for binary vector \( b \) and \( \bar{b} \), there exists a nice relationship between Hamming distance and euclidean distance: \( d_H(\bar{b}, \bar{b}) = \|\bar{b} - \bar{b}\|_2^2 \). The reformulated objective is as follows:

\[
\begin{align*}
\min_{\bar{b}, q, u_1 \in S_b, u_2 \in S_p, u_3 \in R^+} \lambda_1 L_1(\phi(D_1; q); \bar{b}) + \lambda_2 L_2(D_2; \bar{b}), \\
\text{s.t. } \bar{b} = u_1, \quad \bar{b} = u_2, \quad \|\bar{b} - \bar{b}\|_2^2 - k + u_3 = 0,
\end{align*}
\]

where two extra variables \( u_1, u_2 \in \mathbb{R}^+ \) are introduced to split the constraints w.r.t. \( \bar{b} \). Besides, the non-negative slack variable \( u_3 \in \mathbb{R}^+ \) is used to transform \( \|\bar{b} - \bar{b}\|_2^2 - k \leq 0 \) in (2) into \( \|\bar{b} - \bar{b}\|_2^2 - k + u_3 = 0 \). An approximate solution of the above constrained optimization problem can be efficiently obtained by the alternating direction method of multipliers (ADMM) [55].

Following the standard procedure of ADMM, we first present the augmented Lagrangian function of the above problem, as follows:

\[
\begin{align*}
L(\bar{b}, q, u_1, u_2, u_3, z_1, z_2, z_3) &= \lambda_1 L_1(\phi(D_1; q); \bar{b}) + \lambda_2 L_2(D_2; \bar{b}) \\
&+ z_1^T (\bar{b} - u_1) + z_2^T (\bar{b} - u_2) + z_3 (\|\bar{b} - \bar{b}\|_2^2 - k + u_3) \\
&+ c_1 (u_1) + c_2 (u_2) + c_3 (u_3) \\
&+ \rho_1 \|\bar{b} - u_1\|_2^2 + \rho_2 \|\bar{b} - u_2\|_2^2 \\
&+ \rho_3 \|\bar{b} - \bar{b}\|_2^2 - k + u_3)^2,
\end{align*}
\]

where \( z_1, z_2 \in \mathbb{R}^V \) and \( z_3 \in \mathbb{R} \) are dual variables, and \( \rho_1, \rho_2, \rho_3 > 0 \) are penalty factors, which will be specified later. \( c_1 (u_1) = 1_{\{u_1 \in S_b\}}, c_2 (u_2) = 1_{\{u_2 \in S_p\}}, \) and \( c_3 (u_3) = 1_{\{u_3 \in \mathbb{R}^+\}} \) capture the constraints \( S_b, S_p, \) and \( \mathbb{R}^+ \), respectively. The indicator function \( 1_{\{a\}} = 0 \) if \( a \) is true; otherwise, \( 1_{\{a\}} = +\infty \). Based on the augmented Lagrangian function, the primary and dual variables are updated iteratively, with \( r \) indicating the iteration index.

Given \((\bar{b}, q^r, z_1^r, z_2^r, z_3^r))\), Update \((u_1^{r+1}, u_2^{r+1}, u_3^{r+1})\):

\[
\begin{align*}
\begin{cases}
\hat{u}_1^{r+1} = \arg \min_{u_1 \in S_b} \left\{ z_1^T (\bar{b} - u_1) + \frac{\rho_1}{2} \|\bar{b} - u_1\|_2^2 \right\}, \\
\hat{u}_2^{r+1} = \arg \min_{u_2 \in S_p} \left\{ z_2^T (\bar{b} - u_2) + \frac{\rho_2}{2} \|\bar{b} - u_2\|_2^2 \right\}, \\
\hat{u}_3^{r+1} = \arg \min_{u_3 \in \mathbb{R}^+} \left\{ z_3^T (\|\bar{b} - \bar{b}\|^2_2 - k + u_3) + \frac{\rho_3}{2} \|\bar{b} - \bar{b}\|^2_2 - k + u_3 \right\}.
\end{cases}
\end{align*}
\]
\( \hat{a} = a - \frac{1}{2} \) indicates the projection onto the \( \ell_2 \)-sphere constraint \( S_p \). \( \mathcal{P}_{\mathbb{R}^+}(a) = \max(0, a) \) with \( a \in \mathbb{R} \) indicates the projection onto \( \mathbb{R}^+ \).

Given \( (u_1^{r+1}, u_2^{r+1}, u_3^{r+1}, z_1^r, z_2^r, z_3^r) \), update \( b^{r+1} \) and \( q^r \): Although there is no closed-form solution to \( b^{r+1} \), it can be easily updated by the gradient descent method, as \( L_1(\phi(D_1; q), t; b) \) and \( L_2(D_2; b) \) are differentiable w.r.t. \( b \), as follows

\[
\begin{align*}
\frac{\partial L(b, q, u_1^{r+1}, u_2^{r+1}, u_3^{r+1}, z_1^r, z_2^r, z_3^r)}{\partial b} & \bigg|_{b=b^r} \\
\Rightarrow b^{r+1} & \leftarrow b^r - \eta \cdot \frac{\partial L(b, q, u_1^{r+1}, u_2^{r+1}, u_3^{r+1}, z_1^r, z_2^r, z_3^r)}{\partial b},
\end{align*}
\]

where \( \eta > 0 \) denotes the step size. Note that we can run multiple steps of gradient descent in the above update. Both the number of steps and \( \eta \) will be specified in later experiments. Besides, the detailed derivation of \( \frac{\partial L}{\partial b} \) for SSA and TSA can be found in Appendix B, available online.

As mentioned in Section III-B, we suppose that \( \phi \) is differentiable w.r.t. \( q \) and thus \( L_1(\phi(D_1; q), t; b) \) is differentiable w.r.t. \( q \). We can update \( q \) using the gradient descent method as follows

\[
q^{r+1} \leftarrow q^r - \zeta \cdot \frac{\partial L(b^r, q^r, u_1^{r+1}, u_2^{r+1}, u_3^{r+1}, z_1^r, z_2^r, z_3^r)}{\partial q},
\]

where \( \zeta > 0 \) denotes the step size. The update of \( q \) is kept pace with \( b \), i.e., we update \( q \) for one step for each update of \( b \). Note that the update of \( q \) is kept pace with \( b \), using the same backward pass.

Given \( (b^{r+1}, q^{r+1}, u_1^{r+1}, u_2^{r+1}, u_3^{r+1}) \), Update \( (z_1^{r+1}, z_2^{r+1}, z_3^{r+1}) \). The dual variables are updated by the gradient ascent method, as follows

\[
\begin{align*}
z_1^{r+1} &= z_1^r + \rho_1(b^{r+1} - u_1^{r+1}), \\
z_2^{r+1} &= z_2^r + \rho_2(b^{r+1} - u_2^{r+1}), \\
z_3^{r+1} &= z_3^r + \rho_3(||b^{r+1} - u_3^{r+1}||_2 - k + u_5^{r+1}).
\end{align*}
\]

Remarks: 1) Note that since the dual variables \( (u_1^{r+1}, u_2^{r+1}, u_3^{r+1}) \) are updated in parallel, and the primal variables \( (b, q) \) are also updated in parallel, the above algorithm is a two-block ADMM algorithm. We summarize this algorithm in Appendix C, available online. 2) Except for the update of \( b^{r+1} \) and \( q^{r+1} \), other updates are very simple and efficient. The computational cost of the whole algorithm will be analyzed in Section IV-E2. 3) Due to the inexact solutions to \( b^{r+1} \) and \( q^{r+1} \) using gradient descent, the ADMM algorithm gives an approximate solution to Problem (12) and its theoretical convergence cannot be guaranteed. However, as demonstrated in many previous works [55, 56, 57], the inexact two-block ADMM often shows good practical convergence, which is also the case in our later experiments. Besides, the numerical convergence analysis is presented in Section IV-E3. 4) The proper adjustment of \( (\rho_1, \rho_2, \rho_3) \) could accelerate the practical convergence, which will be specified later.

\[a \rightarrow \text{Extension to Full-Precision Networks}\]

To further improve the versatility of our method, we extend it to attack full-precision DNNs whose weights are in 32-bit floating point format. According to IEEE standard [58], for each weight stored as the binary format \( w = [w_{32}; w_{31}; \ldots; w_1] \), the sign bit \( (w_{32}) \) determines the sign of the number, the exponent part uses 8 bits \( (w_{31}; \ldots; w_{24}) \) to encode the exponent bias, and the fraction part uses 23 bits \( (w_{23}; \ldots; w_1) \) to encode the mantissa. Usually, each binary vector \( w \) can be converted to a real number by a function \( h_F(\cdot) \), as follow

\[h_F(w) = \left( -1 \right)^{w_{32}} \cdot 2^{w_{24} - 127} \cdot \left( 1 + \sum_{i=1}^{23} 2^{-i} \cdot w_{24-i} \right),
\]

where \( E = \sum_{i=24}^{31} 2^{i-24} \cdot w_i \) denotes the exponent bias. For attacking the full-precision networks, we replace \( h(\cdot) \) (defined in Section III-A) with \( h_F(\cdot) \) in our general formulation. The sign of the full-precision weight \( s_F \in \{ -1, 1 \} \) is calculated by \( s_F = \left( -1 \right)^{w_{32}} \). If \( w_{32} = 0 \), the weight is positive; otherwise, it is negative.

There are two challenges for extending the proposed optimization method to attack the full-precision networks. First, because \( \partial s_F / \partial w_{32} = \left( -1 \right)^{w_{32}} \ln(-1) \), it is infeasible to update the sign bit \( w_{32} \) with the gradient method using (18). We alternatively use \( s_F = 1 - 2 \cdot w_{32} \) to calculate the sign of the full-precision weight in our implementation. For the function \( s_F = 1 - 2 \cdot w_{32} \), if \( w_{32} = 0 \), \( s_F = 1 \); otherwise, it is \(-1\). Using this alternative function, we can calculate the derivation of \( \partial s_F / \partial w_{32} \) properly. Second, we find that directly optimizing all bits of weights in the full-precision networks is infeasible. This is because modifying the high-order bits in the exponent part of floating point numbers can increase the weights to extremely large values, thus resulting in exploded outputs and unexpected gradients. To address this problem, we propose to set gradients of some bits as zeros when we update the binary parameters \( b \) of the attacked full-precision model. It is implemented by a gradient mask \( M \in \{ 0, 1 \}^V \), whose shape is the same as that of the optimized parameters \( b \):

\[
\frac{\partial L(b, q^r, u_1^{r+1}, u_2^{r+1}, u_3^{r+1}, z_1^r, z_2^r, z_3^r)}{\partial b} \bigg|_{b=b^r} = \nu \cdot M
\]

where \( M \) is created depending on the number of masked high-order bits in the exponent part. Here, we take TSA as an example to introduce how to pre-define the mask \( M \). For TSA, \( b \) corresponds to the reshaped weights of the fully-connected layer \( b^r \in \{ 0, 1 \}^{K \times C \times Q} \), where \( b_{\text{rel,i}}(i \in \{ 31, 30, \ldots, 24 \}) \) encode the exponent bias. We use \( M^b \in \{ 0, 1 \}^{K \times C \times Q} \) to denote the binary mask before reshaping. If we mask \( N_M \in \{ 1, \ldots, 8 \} \) high-order bits in the exponent part, \( M^b_{\text{rel,i}}(i \in \{ 31, 30, \ldots, 32-N_M \}) \) are set as 0 and other elements are 1. Finally, we obtain \( M \in \{ 0, 1 \}^V \) by reshaping \( M^b \). Also, it is easy to be applied to SSA by changing the optimized parameters.

**IV. Experiments**

\[a \rightarrow \text{A. Evaluation Setup}\]

1) **Datasets and Target Models**: We conduct experiments on CIFAR-10 [59] and ImageNet [60]. For SSA and the baseline
TABLE II
RESULTS OF FIVE ATTACK METHODS ACROSS DIFFERENT BIT-WIDTHS AND ARCHITECTURES ON CIFAR-10 AND IMAGENET (BOLD: THE BEST; UNDERLINE: THE SECOND BEST)

| Dataset   | Method | Target Model | PA-ACC (%) | ASR (%) | N_{flip} | Target Model | PA-ACC (%) | ASR (%) | N_{flip} |
|-----------|--------|--------------|------------|---------|----------|--------------|------------|---------|----------|
| CIFAR-10  | Pt     | ResNet 8-bit | 85.0 ± 1.20 | 100.0   | 150.5 ± 1.84 | VGG 8-bit | 84.3 ± 2.10 | 97.9   | 11297.7 ± 830.34 |
|           | T-BFA  | ResNet 8-bit | 87.5 ± 2.22 | 98.7   | 8.91 ± 1.20   |             | 89.8 ± 3.92 | 98.7   | 24.53 ± 1.26   |
|           | PSA    | ResNet 8-bit | 86.3 ± 2.28 | 98.9   | 165.51 ± 3.93 |             | 86.8 ± 2.38 | 98.6   | 253.92 ± 1.02   |
|           | GDA    | ACC: 92.16% | 87.6 ± 3.50 | 99.8   | 28.83 ± 12.50 |             | 85.2 ± 2.26 | 100.0  | 21.54 ± 4.79    |
|           | SSA    | ACC: 92.16% | 88.2 ± 2.46 | 100.0  | 6.87 ± 2.58   |             | 86.2 ± 2.32 | 100.0  | 7.40 ± 2.72    |
| ImageNet  | Pt     | ResNet 4-bit | 84.7 ± 2.74 | 100.0  | 392.4 ± 6.74  | VGG 4-bit | 85.3 ± 2.39 | 94.7   | 2279.52 ± 125.46 |
|           | T-BFA  | ResNet 4-bit | 86.6 ± 2.60 | 97.9   | 8.20 ± 2.26   |             | 88.7 ± 2.62 | 96.2   | 11.23 ± 2.26   |
|           | PSA    | ResNet 4-bit | 87.5 ± 2.34 | 99.4   | 78.33 ± 25.57 |             | 87.8 ± 2.50 | 97.5   | 7.03 ± 25.72   |
|           | GDA    | ACC: 91.90% | 86.2 ± 2.59 | 99.8   | 16.0 ± 2.74   |             | 85.2 ± 2.26 | 100.0  | 10.31 ± 2.37   |
|           | SSA    | ACC: 91.90% | 87.6 ± 2.60 | 100.0  | 6.35 ± 1.69   |             | 85.1 ± 2.33 | 100.0  | 6.56 ± 2.30    |
| CIFAR-10  | Pt     | ResNet 8-bit | 23.9 ± 0.35 | 100.0  | 2704.3 ± 113.34 | VGG 8-bit | 42.0 ± 2.36 | 100.0  | 72965.2 ± 12353.54 |
|           | T-BFA  | ResNet 8-bit | 68.7 ± 0.36 | 79.3   | 24.57 ± 20.03 |             | 73.0 ± 0.12 | 84.5   | 363.74 ± 135.28 |
|           | PSA    | ResNet 8-bit | 67.2 ± 0.15 | 99.7   | 441.31 ± 1.19 |             | 73.2 ± 0.07 | 100.0  | 1030.0 ± 290.30 |
|           | GDA    | ACC: 69.50% | 69.3 ± 0.21 | 100.0  | 18.54 ± 14.34 |             | 73.9 ± 0.02 | 100.0  | 157.05 ± 48.85  |
|           | SSA    | ACC: 69.50% | 69.1 ± 0.08 | 100.0  | 7.37 ± 2.18   |             | 73.5 ± 0.01 | 100.0  | 69.59 ± 34.42   |
| ImageNet  | Pt     | ResNet 4-bit | 12.9 ± 0.31 | 100.0  | 2898.2 ± 132.51 | VGG 4-bit | 71.6 ± 0.15 | 89.5   | 350.33 ± 146.97 |
|           | T-BFA  | ResNet 4-bit | 65.6 ± 0.42 | 80.4   | 24.59 ± 14.97 |             | 71.6 ± 0.09 | 100.0  | 441.32 ± 111.26 |
|           | PSA    | ResNet 4-bit | 66.4 ± 0.21 | 99.9   | 137.53 ± 16.66 |             | 71.3 ± 0.09 | 100.0  | 107.18 ± 29.79  |
|           | GDA    | ACC: 66.77% | 65.9 ± 0.12 | 100.0  | 11.42 ± 3.62  |             | 70.3 ± 0.06 | 100.0  | 95.72 ± 18.32   |
|           | SSA    | ACC: 66.77% | 66.8 ± 0.27 | 100.0  | 7.26 ± 2.39   |             | 70.7 ± 0.01 | 100.0  | 69.72 ± 18.32   |

The mean and standard deviation of PA-ACC and N_{flip} are calculated by attacking the 1,000 images. Our method is denoted as SSA. Note that FSA and GDA are adapted to attack the quantized network in this table.

We attack 1,000 images with a randomly selected target class for each image on both datasets. For TSA and the baseline methods, we select all 10 classes for CIFAR-10 and randomly select 5 classes for ImageNet as the target classes. Besides, for all methods except GDA which does not employ auxiliary samples, we provide 128 and 512 auxiliary samples on CIFAR-10 and ImageNet, respectively, which corresponds to the size of D2 (N2) for SSA and the size of D (N) for TSA. Following the settings in [24], [25], we adopt ResNet [1] and VGG [61] for evaluation. We use ResNet-20 and VGG-16 on CIFAR-10, and ResNet-18 and VGG-16 on ImageNet. In the experiments of quantized networks, we use 4-bit and 8-bit quantized models. We provide more details of datasets and target models in Appendix D, available online.

2) Implementation Details: The implementation details of our method can be found in Appendix D, available online. Specifically, details of SSA in Section 4.2 are in Appendix D.3, available online, details of TSA in Section 4.3 are in Appendix D.4, available online, details of SSA and TSA against full-precision networks are in Appendix D.5, available online, and details of SSA and TSA against MobileNet-V2 and EfficientNet are in Appendix D.6, available online. We also present the ablation studies on hyper-parameters of SSA and TSA in Appendix E, available online.

3) Evaluation Metrics: We adopt three metrics to evaluate the attack performance, i.e., the post attack accuracy (PA-ACC), the attack success rate (ASR), and the number of bit-flips (N_{flip}). PA-ACC denotes the post attack accuracy on the original validation set. ASR is defined as the ratio of attacked samples that are successfully attacked into the target class. To be specific, we calculate ASR using all 1,000 attacked samples for the methods in Section IV-B and all testing samples with the trigger for the methods in Section IV-C. Therefore, we can calculate an ASR after 1,000 attacks for the methods in Section IV-B and obtain an ASR after an attack for the methods in Section IV-C. N_{flip} is the number of bit-flips required for an attack, which is an important metric to reflect whether the attack is efficient and practical. A better attack performance corresponds to a higher PA-ACC and ASR, while a lower N_{flip}. Besides, we show the accuracy of the original model, denoted as ACC.

4) Defense Methods: Besides attacking the standard training models, we test all attacks against two defense methods: piecewise-clustering [37] and larger model capacity [25], [37]. Their details can be found in Appendix D.7, available online. We conduct experiments with the 8-bit quantized ResNet on CIFAR-10 and ImageNet. Besides the three metrics in Section IV-A3, we present the number of increased N_{flip} compared to the model without defense (i.e., results in Tables II and IV), denoted as ΔN_{flip}.

B. Results of SSA

1) Baseline Methods: We compare our SSA with GDA [31], FSA [21], and T-BFA [25]. Since GDA [31] and FSA [21] are originally designed for attacking the full-precision networks, we adapt these two methods to attack the quantized networks by applying quantization-aware training [62]. We adopt the L0-norm for FSA [31] and modification compression for GDA [21] to reduce the number of the modified parameters. Among three types of T-BFA [25], we compare to the most comparable one: the 1-to-1 stealthy attack scheme. The purpose of this attack scheme is to misclassify samples of a single source class into the target class while maintaining the prediction accuracy of other samples. Besides, we take the fine-tuning (FT) of the last fully-connected layer using the objective defined in (6) without considering the constraints as a basic attack and present its results.

2) Main Results: Results on CIFAR-10: The results of all methods on CIFAR-10 are shown in Table II. Our method achieves a 100% ASR with the fewest N_{flip} for all the bit-widths and architectures. FT modifies the maximum number of bits among all methods since there is no limitation of parameter modifications. Due to the absence of large-scale training data, the PA-ACC of FT is also poor. These results indicate that
TABLE III
RESULTS OF ALL ATTACK METHODS AGAINST THE MODELS WITH DEFENSE ON CIFAR-10 AND ImageNet (Bold: The Best; Underline: The Second Best)

| Defense       | Dataset | Method  | ACC (%) | PA-ACC (%) | ASR (%) | N_{flip} | ΔN_{flip} |
|---------------|---------|---------|---------|------------|---------|----------|-----------|
|               |         | T-BFA   | 91.01   | 88.56 ± 2.24 | 99.5     | 1995±249 | 993 ± 389 |
|               |         | T-RE    | 85.38 ± 2.19 | 83.5 ± 2.42 | 99.6     | 234 ± 25.9 | 234 ± 25.9 |
|               |         | FSA     | 85.82 ± 2.51 | 86.5 ± 2.42 | 99.9     | 248 ± 12.53 | 60 ± 60 |
|               |         | GDA     | 84.72 ± 2.77 | 84.7 ± 2.19 | 100.0    | 29.7 ± 14.29 | 25.93 |
|               |         | SSA     | 87.92 ± 2.74 | 88.7 ± 2.74 | 100.0    | 18.9 ± 7.31 | 13.36 |
|               |         | TSA     | 84.73 ± 2.37 | 84.73 ± 2.37 | 99.9     | 78.9 ± 78.9 | 78.9 ± 78.9 |
|               |         | T-BFA   | 82.82 ± 2.87 | 82.81 ± 2.87 | 90.1     | 273 ± 19.29 | 248 ± 99 |
|               |         | FSA     | 83.81 ± 2.87 | 83.81 ± 2.87 | 99.5     | 729 ± 91.83 | 288 ± 73 |
|               |         | GDA     | 86.81 ± 2.87 | 86.81 ± 2.87 | 100.0    | 107 ± 31.35 | 99.99 |
|               |         | SSA     | 87.82 ± 2.87 | 87.82 ± 2.87 | 100.0    | 51.1 ± 43.57 | 43.74 |
|               |         | T-BFA   | 100.00    | 100.00     | 100.0    | 0.0      | 0.0 |
|               |         | T-RE    | 90.75 ± 2.37 | 90.75 ± 2.37 | 98.5     | 271 ± 21.83 | 85.76 |
|               |         | FSA     | 90.75 ± 2.37 | 90.75 ± 2.37 | 98.5     | 271 ± 21.83 | 85.76 |
|               |         | GDA     | 90.75 ± 2.37 | 90.75 ± 2.37 | 98.5     | 271 ± 21.83 | 85.76 |
|               |         | SSA     | 90.75 ± 2.37 | 90.75 ± 2.37 | 98.5     | 271 ± 21.83 | 85.76 |
|               |         | TS     | 94.29    | 94.29     | 99.9     | 79.9 ± 79.9 | 79.9 ± 79.9 |
|               |         | T-BFA   | 85.8 ± 2.87 | 85.8 ± 2.87 | 99.7     | 171 ± 14.44 | 6.02 |
|               |         | FSA     | 90.8 ± 2.37 | 90.8 ± 2.37 | 99.7     | 360 ± 21.83 | 22.13 |
|               |         | T-RE    | 90.8 ± 2.37 | 90.8 ± 2.37 | 99.7     | 360 ± 21.83 | 22.13 |
|               |         | SSA     | 90.8 ± 2.37 | 90.8 ± 2.37 | 99.7     | 360 ± 21.83 | 22.13 |
|               |         | T-BFA   | 71.30 ± 2.87 | 71.30 ± 2.87 | 88.9     | 60.4 ± 6.25 | 53.8 |
|               |         | FSA     | 71.30 ± 2.87 | 71.30 ± 2.87 | 88.9     | 60.4 ± 6.25 | 53.8 |
|               |         | T-RE    | 71.30 ± 2.87 | 71.30 ± 2.87 | 88.9     | 60.4 ± 6.25 | 53.8 |
|               |         | SSA     | 71.30 ± 2.87 | 71.30 ± 2.87 | 88.9     | 60.4 ± 6.25 | 53.8 |

The mean and standard deviation of PA-ACC and N_{flip} are calculated by attacking the 1,000 images. Our method is denoted as SSA. ΔN_{flip} denotes the increased N_{flip} compared to the corresponding result in Table II. Note that FSA and GDA are adapted to attack the quantized network in this table.

Results on CIFAR-10: The results on CIFAR-10 are shown in Table III. It shows the competitive performance of GDA compared to other methods. However, our method obtains the highest PA-ACC, the fewest bit-flips (less than 8), and a 100% ASR in attacking ResNet. For VGG, our method also achieves a 100% ASR with the fewest N_{flip} for both bit-widths. The N_{flip} results of our method are mainly attributed to the cardinality constraint on the number of bit-flips. Moreover, for our method, the average PA-ACC degradation over four cases on ImageNet is only 0.06%, which demonstrates the stealthiness of our attack. The results show that our method is more robust against the bit-flip based attack. However, our method still achieves 100% ASRs with the fewest N_{flip} and ΔN_{flip}. Moreover, when comparing the two defense methods, we find that piece-wise clustering performs better than the model with a larger capacity in terms of ΔN_{flip}. However, piece-wise clustering training also causes the accuracy decrease of the original model (e.g., from 92.16% to 91.01% on CIFAR-10).

Resistance to Larger Model Capacity: The results are presented in Table III. We observe that all methods require more bit-flips to attack the model with the 2 × 2 width. To some extent, it demonstrates that the wider network with the same architecture is more robust against the bit-flip based attack. However, our method still achieves 100% ASRs with the fewest N_{flip} and ΔN_{flip}. Moreover, when comparing the defense models, we find that piece-wise clustering performs better than the model with a larger capacity in terms of ΔN_{flip}. However, piece-wise clustering training also causes the accuracy decrease of the original model (e.g., from 92.16% to 91.01% on CIFAR-10).

Visualization of Decision Boundary: To further compare FSA and GDA with our method, we visualize the decision boundaries of the original and the attacked models in Fig. 2. We adopt a four-layer Multi-Layer Perceptron trained with the simulated 2-D Blob dataset from 4 classes. The original decision boundary indicates that the original model classifies all data points almost perfectly. The attacked sample is classified into Class 3 by all methods. Visually, GDA modifies the decision boundary drastically, especially for Class 0. However, our method modifies the decision boundary mainly around the attacked sample. Although FSA is comparable to ours visually, it flips 10 × bits than GDA and SSA. In the terms of the numerical results, SSA achieves the best PA-ACC and the fewest N_{flip}. This finding verifies that our method can achieve a successful attack even only by tweaking the original classifier.

C. Results of TSA

1) Baseline Methods: We compare TSA with TBT [24], which consists of trigger generation and weight bits identification. We also present the results of fine-tuning (FT) the last fully-connected layer and optimizing the trigger with the objective defined in (10), without considering the constraints. We keep the same trigger design for all methods, especially for Class 0. However, our method modifies the decision boundary mainly around the attacked sample. Although FSA is comparable to ours visually, it flips 10 × bits than GDA and SSA. In the terms of the numerical results, SSA achieves the best PA-ACC and the fewest N_{flip}. This finding verifies that our method can achieve a successful attack even only by tweaking the original classifier.

2) Main Results: Results on CIFAR-10: The results of TSA and the compared methods are shown in Table IV. It shows that FT achieves a higher ASR for VGG, but the accuracy on original...
TABLE IV
RESULTS OF THREE ATTACK METHODS ACROSS DIFFERENT BIT-WIDTHS AND ARCHITECTURES ON CIFAR-10 AND IMAGENET (BOLD: THE BEST)

| Dataset | Method | Target Model | PA-ACC (%) | ASR (%) | N_flip | Target Model | PA-ACC (%) | ASR (%) | N_flip |
|---------|--------|--------------|------------|---------|--------|--------------|------------|---------|--------|
| CIFAR-10 | FT     | ResNet 8-bit ACC: 92.16% | 84.23±0.47 | 84.25±0.10 | 55.4±9.75 | VGG 8-bit ACC: 93.20% | 97.95±0.19 | 97.92±0.13 | 2130.7±92.40 |
|         | TBT    | ResNet 8-bit ACC: 92.16% | 87.52±0.94 | 86.50±0.17 | 63.7±4.88 | VGG 8-bit ACC: 93.20% | 97.95±0.19 | 97.92±0.13 | 2130.7±92.40 |
| TBT    |       | ResNet 4-bit ACC: 91.90% | 88.09±0.14 | 96.06±1.25 | 4.5±1.09 | VGG 8-bit ACC: 93.20% | 97.95±0.19 | 97.92±0.13 | 2130.7±92.40 |
| ImageNet | FT    | ResNet 8-bit ACC: 69.50% | 69.24±0.05 | 75.38±4.90 | 96.5±13.02 | VGG 8-bit ACC: 69.24±0.05 | 75.38±4.90 | 96.5±13.02 | 2130.7±92.40 |
| TBT    |       | ResNet 8-bit ACC: 69.50% | 65.99±0.10 | 74.99±0.04 | 60.4±8.49 | VGG 8-bit ACC: 69.24±0.05 | 75.38±4.90 | 96.5±13.02 | 2130.7±92.40 |
| TSA    |       | ResNet 8-bit ACC: 69.50% | 65.49±0.10 | 95.63±1.23 | 3.4±1.82 | VGG 8-bit ACC: 69.24±0.05 | 75.38±4.90 | 96.5±13.02 | 2130.7±92.40 |

The mean and standard deviation of PA-ACC, ASR, and N_flip are calculated by attacking into 10 and 5 target classes on CIFAR-10 and ImageNet, respectively. Our method is denoted as TBA. SSA can achieve 100% ASRs with the least number of bit-flips and the competitive PA-ACC in all cases. All results verify the superiority of SSA compared with FT and TBT.

D. Results on Full-Precision Networks

Here, we conduct experiments on full-precision ResNet and VGG on CIFAR-10 and ImageNet. We compare FT, SSA, and GDA with our SSA in Table VI, and FT with our SSA in Table VII. Table VI shows that all methods can achieve high ASRs, indicating that full-precision models are also vulnerable to the weight attack. Similar to the observation from Table II, SSA can achieve 100% ASRs with the least number of bit-flips and the competitive PA-ACC in all cases. Table VII shows that for FT, the ASR is higher but the number of bit-flips is unacceptable. SSA requires a very small number of bit-flips to achieve high ASRs (>92%) and performs better in terms of PA-ACC. These results demonstrate that based on the techniques described in Section III-F, our method can achieve superior performance in attacking full-precision models. The mean and standard deviation of PA-ACC, ASR, and N_flip are calculated by attacking into 10 and 5 target classes on CIFAR-10 and ImageNet, respectively. Our method is denoted as TBA. SSA can achieve 100% ASRs with the least number of bit-flips and the competitive PA-ACC in all cases. All results verify the superiority of SSA compared with FT and TBT.

E. A Closer Look at the Proposed Method

1) Effect of the Target Class: The results of SSA and TBA with different target classes are shown in Appendix G, available online.
We analyze the complexity of SSA, indicating that the overall computational cost of SSA is $O(T_{outer} \cdot T_{inner} \cdot |C|)$, where $T_{outer}$ is the number of outer iterations and $T_{inner}$ is the number of gradient steps in updating $\hat{b}^{r+1}$, making the optimization process feasible within an acceptable time.

2) Complexity Analysis: We analyze the complexity of SSA and TSA in detail in Appendix H, available online. According to the analysis, the overall computational cost of SSA is $O(T_{outer} \cdot T_{inner} \cdot [2(N_2 + 1)CQ \cdot (C + 1)])$, with $T_{outer}$ being the iteration of the outer algorithm and $T_{inner}$ indicating the number of gradient steps in updating $\hat{b}^{r+1}$. As shown in Section IV-E3, SSA always converges very fast, thus $T_{outer}$ is not very large. Also, $T_{inner}$ is set to 5 in Section IV-B. Thus, SSA can be optimized efficiently. For TSA, the cost of updating $\hat{b}$ per iteration is $O(2KNCQ)$ in the gradient descent. The cost of the forward process and computing the gradients of $L$ w.r.t. $q$ is the same as that of the standard model training. Moreover, due to the very small auxiliary sample set, TSA could be optimized within an acceptable time.

3) Numerical Convergence Analysis: We present the numerical convergence of SSA on ImageNet in Fig. 3. Note that $||\hat{b} - u_1||_2$ and $||\hat{b} - u_2||_2$ characterize the degree of satisfaction of the box and $L_2$-sphere constraint, respectively. $\lambda_1L_1 + \lambda_2L_2$ means the attack performance to some extent, where a lower value corresponds to a better attack. We can observe that the optimization stops when meeting $||\hat{b} - u_1||_2 \leq 10^{-4}$ and $||\hat{b} - u_2||_2 \leq 10^{-4}$ and do not exceed the maximum number of iterations, indicating the fast convergence of our proposed method in practice. We also provide detailed analysis in Appendix I, available online.

4) A Further Discussion About the Stealthiness: We consider a new stealthiness loss and another definition of stealthiness in this section. Specifically, we utilize the below loss to encourage the predictions of the attacked model to be the same as those of the original model on samples except the attacked one(s)

$$L_2(D_2; \hat{b}) = \sum_{(x_i, y_i) \in D_2} \ell'(f(x_i; \hat{b}), f(x_i; b)),$$

where $\ell'(\cdot, \cdot)$ calculates the Kullback-Leibler (KL) divergence. Besides, we use a new metric, i.e., Inconsistency Rate (IR) to measure the difference between the predictions of the attacked model and those of the original model.

$$IR = 100\% \times \sum_{(x_i, y_i) \in D_T} \frac{1}{|D_T|}$$

(21)

where $I_{ij}$ is the indicator function and $|D_T|$ denotes the size of the test set. Note that a better attack stealthiness corresponds to a lower IR.

We show the results of two forms of $L_2$ ((3) and (20)) in Table VIII with the proposed metric IR. We can observe from the table that IRs are very low in all cases (<6% on CIFAR-10 and <0.6% on ImageNet), indicating the predictions of the original model and the attacked model are very similar. The low IR of using (3) is because these victim models are originally obtained by optimizing the cross entropy loss and have small loss values. On CIFAR-10, we can obtain lower IRs using (20). However, the ASR of the methods with the loss defined by (21) is lower, especially for TSA. The reason may be that encouraging the predictions of the original model and the attacked model to be the same implies a stronger restriction for the algorithm to identify the critical bits for attacking, compared with using (3).

![Table VI: Results of Four Attack Methods Across Different Full-Precision Networks on CIFAR-10 and ImageNet (Bold: The Best; Underline: The Second Best)](image)

| Dataset | Method | Target Model | PA-ACC (%) | ASR (%) | $N_{ep}$ |
|---------|--------|--------------|------------|--------|---------|
| CIFAR-10 | PT | ResNet | 84.87 ± 0.24 | 100.00 | 77/14.83 ± 10.13 |
| | GDA | 86.66 ± 0.31 | 100.00 | 854.54 ± 107.54 |
| | SSA | 88.65 ± 0.03 | 100.00 | 80.04 ± 33.87 |
| | VGG | 94.68 ± 0.03 | 100.00 | 530.32 ± 101.64 |
| | GDA | 88.35 ± 0.24 | 99.5 | 3597.99 ± 109.07 |
| | SSA | 85.55 ± 0.12 | 100.00 | 6.98 ± 3.38 |

The mean and standard deviation of PA-ACC and $N_{ep}$ are calculated by attacking the 1,000 images. Our method is denoted as SSA.

![Table VII: Results of Two Attack Methods Across Different Full-Precision Networks on CIFAR-10 and ImageNet (Bold: The Best)](image)

| Dataset | Method | Target Model | PA-ACC (%) | ASR (%) | $N_{ep}$ |
|---------|--------|--------------|------------|--------|---------|
| CIFAR-10 | PT | ResNet | 83.12 ± 0.32 | 97.50 ± 1.38 | 8154.33 ± 75.84 |
| | TSA | 87.17 ± 0.39 | 95.43 ± 3.33 | 7.3 ± 0.04 |
| | VGG | 82.90 ± 0.32 | 93.32 ± 14.48 | 5783.52 ± 292.47 |
| | GDA | 86.89 ± 0.43 | 97.09 ± 1.41 | 55.4 ± 2.26 |
| | SSA | 78.92 ± 0.43 | 95.32 ± 1.46 | 2.4 ± 1.00 |
| | VGG | 73.26 ± 0.16 | 93.43 ± 1.27 | 21.4 ± 1.12 |
| | GDA | 72.14 ± 0.15 | 90.00 ± 0.00 | 51.7 ± 2.12 |
| | SSA | 72.14 ± 0.15 | 90.00 ± 0.00 | 51.7 ± 2.12 |

The mean and standard deviation of PA-ACC, ASR, and $N_{ep}$ are calculated by attacking into 10 and 5 target classes on CIFAR-10 and ImageNet, respectively. Our method is denoted as TSA.

In all cases, SSA achieves a 100.0% ASR and about 88% PA-ACC with a few bit-flips. It verifies that the target class has little effect on the attack performance of SSA. For TSA, an interesting finding is that the attack performance changes a lot across target classes, which is different from SSA. Therefore, the difficulty of attacking depends on the target class. For example, SSA achieves a relatively high PA-ACC and ASR with only 2 b-flips in attacking into class 2; the PA-ACC is only 78.30% with the highest bit-flips in attacking into class 5.
Eq. (3) encourages the attacked model classifies samples except the attacked one(s) correctly, while Eq. (20) encourage the predictions of the attacked model are the same as those of the original model.

| Method  | Dataset  | $L_2$ | ACC (%) | PA-ACC (%) | ASR (%) | N_{opt} | IR |
|---------|----------|-------|---------|------------|---------|--------|----|
| SSA     | CIFAR-10 | Eq. (3) | 92.16   | 88.29 ± 2.64 | 100.0   | 3.57 ± 0.31 | 5.96 ± 2.85 |
|         | ImageNet | Eq. (3) | 69.50   | 65.61 ± 0.05 | 100.0   | 7.97 ± 0.24 | 0.00 ± 0.24 |
| TFA     | CIFAR-10 | Eq. (3) | 92.16   | 88.98 ± 1.82 | 96.56 ± 2.23 | 4.07 ± 0.55 | 4.97 ± 0.55 |
|         | ImageNet | Eq. (3) | 69.50   | 69.36 ± 0.09 | 95.99 ± 5.12 | 8.38 ± 0.82 | 5.51 ± 1.59 |

The mean and standard deviation of PA-ACC and N_{opt} are calculated by attacking the 1,000 images. Our method is denoted as SSA.

| Method  | Target Model | PA-ACC (%) | ASR (%) | N_{opt} |
|---------|--------------|------------|---------|--------|
| FT      | MobileNetV2   | 73.05 ± 2.15 | 100.0   | 28.42 ± 1.56 |
| T-BFA   | MobileNetV2   | 70.46 ± 0.24 | 100.0   | 21.30 ± 4.50 |
| T-FSA   | MobileNetV2   | 70.02 ± 0.17 | 100.0   | 8.15 ± 0.15 |
| SSA     | MobileNetV2   | 70.01 ± 0.15 | 100.0   | 28.42 ± 1.56 |
| FT      | EfficientNet  | 73.54 ± 0.13 | 100.0   | 27.95 ± 1.52 |
| T-BFA   | EfficientNet  | 73.75 ± 0.12 | 100.0   | 22.05 ± 1.55 |
| T-FSA   | EfficientNet  | 73.75 ± 0.12 | 100.0   | 22.05 ± 1.55 |
| SSA     | EfficientNet  | 73.75 ± 0.12 | 100.0   | 22.05 ± 1.55 |

The mean and standard deviation of PA-ACC and N_{opt} are calculated by attacking into 5 target classes. Our method is denoted as TFA.

Therefore, how to ensure the predictions of the attacked model are unchanged and achieve a high ASR is worthy of further exploration.

5) Results on More Architectures: Here, we test all attack methods on 8-bit quantized MobileNetV2 (width multiplier 1) [63] and EfficientNet-B0 [64] on ImageNet. The results are shown in Tables IX and X. They show that both MobileNetV2 and EfficientNet are vulnerable to the bit-flip based weight attack. Similar to the observation on ResNet and VGG in Table II, SSA achieves 100% ASRs with a low number of bit-flips (<30) as shown in Table IX. The accuracy degradation of SSA is only 0.16% and 0.02% for MobileNetV2 and EfficientNet, respectively. Table X shows that baseline methods cannot obtain an ASR greater than 90%, while TFA achieves high ASRs (>95%) with the least number of bit-flips. These results demonstrate that SSA and TFA are effective in attacking MobileNetV2 and EfficientNet, and have the potential to be used to attack diverse architectures.

6) Effect of the Search Space: We study the effect of the search space for our method in this section. First, we allowed flipping bits in all weights of the attacked model. However, we found that the optimization algorithm fails in this case, because the search space is too large. For example, the number of candidate bits of the original SSA is 1,024 while the number of bits in all weights is 2,146,688 for the 8-bit quantized ResNet on CIFAR-10.

Here, we take a small step toward a larger search space. Specifically, we extend SSA to optimize all bits in the last fully-connected layer while the original SSA modifies weights connected to the target class in the last fully-connected layer. For TFA, we optimize not only the last fully-connected layer as that in the original version but also the last convolutional layer. Besides following the stopping criteria in Sections IV-B and IV-C, to investigate whether increasing search spaces increases the attack run-time, we also allow the methods with larger search spaces to be optimized with more iterations. Specifically, we increase the maximum number of iterations to 5,000 and set another stopping criterion as $\|\hat{b} - u_1\|_2^2 < 10^{-5}$ and $\|\hat{b} - u_2\|_2^2 < 10^{-6}$. The results are shown in Table XI.

We can see that the methods with larger search spaces obtain inferior performance with the same stopping criteria as our original methods. When using more iterations, their performance is comparable with our original methods, indicating that increasing search spaces increases the attack run-time. Furthermore, we allow more iterations (e.g., the maximum number of iterations is 7,000 and another stopping criterion is $\|\hat{b} - u_1\|_2^2 < 10^{-6}$ and $\|\hat{b} - u_2\|_2^2 < 10^{-6}$) for the methods with larger search space and find that it can not significantly improve their attack performance further. Accordingly, for our methods, constraining the search space simplifies the optimization problem and results in faster convergence. Besides, for TFA, an interesting observation is that almost all flipped bits are in the last fully-connected layer, even though the last convolutional layer is included. It demonstrates that the proposed search spaces are reasonable choices for our methods. We also believe that allowing flipping bits in all weights is a challenging but valuable problem and worthy of future investigation.

V. CONCLUSION AND FUTURE WORK

In this article, we have explored a novel attack paradigm that the weights of a deployed DNN can be slightly changed via bit flipping in the memory to achieve some malicious purposes. First, we present the general formulation considering the attack

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
effectiveness and stealthiness. Based on it, we propose two special types of attack: SSA and TSA. Furthermore, we solve the binary optimization (SSA) and the MIP problem (TSA) with an effective and efficient continuous algorithm. Since the critical bits are determined through optimization, SSA and TSA can achieve the attack goals by flipping a few bits under different settings.

Future studies of the bit-flip based attack include, but are not limited to: 1) Specifying our framework as other types of attack with different malicious purposes. For example, it may be extended to the untargeted attack [23] where all samples are misclassified into any incorrect class, and the all-to-all attack [25] where the target class of the attacked sample depends on its ground-truth label; 2) Extending it to the different tasks, e.g., face recognition; 3) Exploring more strict settings than the white-box one; 4) Using our method to evaluate the parameter robustness of models with different architectures, training strategies, etc., due to its least number of bit-flips; 5) Allowing flipping bits in all weights of the attacked model. We hope that this work can draw more attention to the security of the deployed DNNs.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[2] B. Zhang, D. Xiong, and J. Su, “Neural machine translation with deep attention,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 1, pp. 154–163, Jan. 2020.
[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” J. Mach. Learn. Res., vol. 17, no. 1, pp. 1334–1373, 2016.
[4] C. Szegedy et al., “Intriguing properties of neural networks,” in Proc. Int. Conf. Learn. Representations, 2014, pp. 1–10.
[5] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, “BadNets: Evaluating backdooring attacks on deep neural networks,” IEEE Access, vol. 7, pp. 47230–47244, 2019.
[6] Z. Li, D. Gong, X. Li, and D. Tao, “Learning compact feature descriptor and adaptive matching framework for face recognition,” IEEE Trans. Image Process., vol. 24, no. 9, pp. 2736–2745, Sep. 2015.
[7] Y. Dong et al., “Efficient decision-based black-box adversarial attacks on face recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 7706–7714.
[8] H. Wang et al., “CosFace: Large margin cosine loss for deep face recognition,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 5265–5274.
[9] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane, “Adversarial attacks on medical machine learning,” Science, vol. 363, no. 6433, pp. 1287–1289, 2019.
[10] C. J. Kelly, A. Karthikesalingam, M. Suleyman, G. Corrado, and D. King, “Draw more attention to the security of the deployed DNNs,” in Proc. Int. Conf. Learn. Representations, 2019, pp. 1211–1221.
[11] A. S. Rakin, Z. He, and D. Fan, “TBT: Targeted neural network attack with bit trojan,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 13195–13204.
[12] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, “T-BFA: Targeted bit-flip adversarial weight attack,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 11, pp. 7928–7939, Nov. 2022.
[13] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial patch,” in Proc. Workshop Mach. Learn. Comput. Secur., 2017, pp. 1–6.
[14] B. Wu and B. Ghanem, “Adam: A versatile framework for integer programming,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 7, pp. 1695–1708, Jul. 2019.
[15] R. Glowinski and A. Marroco, “Sur l’approximation, par éléments finis d’ordre un, de la résolution, par pénalisation-dualité, d’une classe de problèmes de dirichlet non linéaires,” ESAIM: Math. Modelling Numer. Anal.–Modélisation Mathématique et Analyse Numérique, vol. 9, no. R2, pp. 41–76, 1975.
[16] D. Gabay and B. Mercier, “A dual algorithm for the solution of nonlinear variational problems via finite element approximation,” Comput. Math. Appl., vol. 2, no. 1, pp. 17–40, 1976.
[17] J. Bai, B. Wu, Y. Zhang, Y. Li, Z. Li, and S. T. Xia, “Targeted attack against deep neural networks via flipping limited weight bits,” in Proc. Int. Conf. Learn. Representations, 2021, pp. 1–12.
[18] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault injection attack on deep neural network,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des., 2017, pp. 131–138.
[19] Y. Liu et al., “Trojaning attack on neural networks,” in Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp., 2018, pp. 1–15.
[20] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, “Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks,” in Proc. 28th USENIX Conf. Secur. Symp., 2019, pp. 497–514.
[21] B. Wu, L. Liu, Z. Zhu, Q. Liu, Z. He, and S. Lyu, “Adversarial machine learning: A systematic survey of backdoor attack, weight attack and adversarial example,” 2023, arXiv:2302.09457.
[22] T.-W. Weng, P. Zhao, S. Liu, P.-Y. Chen, X. Lin, and L. Daniel, “Towards certificated model robustness against weight perturbations,” in Proc. AAAI Conf. Artif. Intell., 2022, pp. 6356–6363.
[23] Y. Ji, X. Zhang, S. J. X. Luo, and T. Wang, “Model-reuse attacks on deep learning systems,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2018, pp. 349–363.
[24] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, “Defending and harnessing the bit-flip based adversarial weight attack,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 14083–14091.
[25] D. Stutz, N. Chandra, M. Hein, and B. Schiele, “Bit error robustness for energy-efficient dnn accelerators,” in Proc. Mach. Learn. Syst., 2021, pp. 596–598.
[26] D. Stutz, N. Chandra, M. Hein, and B. Schiele, “Random and adversarial bit error robustness: Energy-efficient and secure DNN accelerators,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 3, pp. 3632–3647, Mar. 2023.
[27] J. Li et al., “Defending bit-flip attack through DNN weight reconstruction,” in Proc. 57th ACM/EDAC/IEEE Des. Autom. Conf., 2020, Art. no. 173.
[41] Z. He, T. Zhang, and R. Lee, “Sensitive-sample fingerprinting of deep neural networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 4724–4732.

[42] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, “RADAR: Run-time adversarial weight attack detection and accuracy recovery,” 2021, arXiv:2101.08254.

[43] H. Guan, L. Ning, Z. Lin, X. Shen, H. Zhou, and S.-H. Lim, “In-place zero-space memory protection for CNN,” in Proc. 33rd Int. Conf. Neural Inf. Process. Syst., 2019, Art. no. 515.

[44] K. Huang, P. H. Siegel, and A. A. Jiang, “Functional error correction for reliable neural networks,” in Proc. IEEE Int. Symp. Inf. Theory, 2020, pp. 2694–2699.

[45] S. Migacz, “8-bit inference with tensort,” in Proc. GPU Technol. Conf., 2017, pp. 1–41.

[46] B. Wu et al., “Backdoorbench: A comprehensive benchmark of backdoor learning,” in Proc. Adv. Neural Inf. Process. Syst., 2022, pp. 10546–10559.

[47] Y. Xiao, X. Zhang, Y. Zhang, and B. Teodorescu, “One bit flips, one cloud flops: Cross-VMM row hammer attacks and privilege escalation,” in Proc. 25th USENIX Conf. Secur. Symp., 2016, pp. 19–35.

[48] F. Zhang et al., “Persistent fault analysis on block ciphers,” IACR Trans. Cryptographic Hardware Embedded Syst., vol. 2018, pp. 150–172, 2018.

[49] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, “Throwhammer: Rowhammer attacks over the network and defenses,” in Proc. USENIX Conf. Usenix Annu. Tech. Conf., 2018, pp. 213–225.

[50] N. Herath and A. Fogh, “These are not your grand daddys CPU performance counters–CPU hardware performance counters for security,” in Proc. Black Hat Brief., 2015, pp. 1–89.

[51] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, “Compressing convolutional neural networks via factorized convolutional filters,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 3972–3981.

[52] A. Bibi, B. Wu, and B. Ghanem, “Constrained k-means with general pairwise and cardinality constraints,” 2019, arXiv:1907.10410.

[53] B. Wu, L. Shen, T. Zhang, and B. Ghanem, “MAP inference via l2-sphere linear program reformulation,” Int. J. Comput. Vis., vol. 128, pp. 1913–1936, 2020.

[54] Y. Fan et al., “Sparse adversarial attack via perturbation factorization,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 35–50.

[55] S. Boyd, N. Parikh, and E. Chu, Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Boston, MA, USA: Now, 2011.

[56] E. G. Gol’shtein and N. Tret’yakov, “Modified lagrangians in convex optimization. I,” Point-to-Set Maps and Mathematical Programming, Berlin, Germany: Springer, 1979, pp. 86–97.

[57] J. Eckstein and D. P. Bertsekas, “On the douglas–rachford splitting method and the proximal point algorithm for maximal monotone operators,” Math. Program., vol. 55, no. 1-3, pp. 293–318, 1992.

[58] IEEE standard for floating-point arithmetic, IEEE Std 754–2019 (Revision of IEEE 754-2008), pp. 1–84, 2019.

[59] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” Technical report, vol. 1, pp. 1–60, 2009.

[60] O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015.

[61] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015, pp. 1–10.

[62] B. Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2704–2713.

[63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetv2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.

[64] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6105–6114.