Concurrent Weight Encoding-based Detection for Bit-Flip Attack on Neural Network Accelerators

Qi Liu
Lehigh University
qil219@lehigh.edu

Wujie Wen
Lehigh University
wuw219@lehigh.edu

Yanzhi Wang
Northeastern University
yanz.wang@northeastern.edu

Abstract

The recent revealed Bit-Flip Attack (BFA) against deep neural networks (DNNs) is highly concerning, as it can completely mislead the inference of quantized DNNs by only flipping a few weight bits in hardware memories through manners like DRAM rowhammer. A key question before applying any BFA mitigation solutions, such as retraining or model reloading, is how to quickly and accurately detect such an attack without impacting the normal inference. In this paper, we propose a weight encoding-based framework to concurrently detect BFA by leveraging the spatial locality of bit flipping in BFA and a fast encoding of sensitive weights only. Extensive experimental results show that our method can accurately differentiate the malicious fault models under BFA and the random bit flipping that could also occur in weight memories but does not impact accuracy as that of BFA, with very low overhead across various DNNs on both CIFAR-10 and ImageNet datasets. To the best of our knowledge, this is the first real-time detection framework for BFA attack against quantized DNNs that are widely deployed in hardware accelerators.

ACM Reference Format:
Qi Liu, Wujie Wen, and Yanzhi Wang. 2020. Concurrent Weight Encoding-based Detection for Bit-Flip Attack on Neural Network Accelerators. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3400302.3415726

1 Introduction

Deep neural networks (DNNs) have been widely applied in many real-world applications such as speech recognition, robotics, and self-driving vehicles. However, it also raises the ever-increasing security concerns. Aside from existing adversarial, poisoning and trojan attacks against DNNs [1, 3, 15], studies have also revealed that DNNs deployed on hardware engines are also highly vulnerable to various types of fault injection attacks by distorting weight parameters or bits stored in memories [5, 9, 16]. For example, simply flipping the exponential bits of the 32-bit floating-point weights through DRAM Row-hammer Attack (RHA) can completely malfunction DNNs [5, 8]. Hardware-favorable quantized DNNs can be immune to aforementioned bit attack [8], unfortunately, the most recent Bit-Flip Attack (BFA) shows that the accuracy of quantized models can be degraded to the level close to a random guess by only flipping a few most vulnerable weight bits stored in DRAM RHA [4, 12], e.g. only 13 bit-flips out of 93 million bits through a progressive bit search (PBS) algorithm [17]. This prompts an immediate need for countermeasures against such a strong attack which could happen to any quantized DNNs and distort the inference running on hardware accelerators immediately.

There exist a few proposals to address the emerging BFA. The first is to eliminate the hardware exploits of BFA, that is, overcoming DRAM Rowhammer vulnerability through probabilistic adjacent row activation (PARA) [12] and error correction code (ECC) memory [4, 18]. However, these solutions either incur complicated memory controller changes or can be easily bypassed by new attack methods like ECCploit [2]. The second is to enhance the model's resistance against BFA through attack-specific training from scratch or retraining-based model reconstruction, so as to increase the number of needed bit-flips to achieve an effective attack [7, 11]. However, these methods cannot eliminate BFA after model deployment and their achievable effectiveness on large and complicated networks and datasets (e.g. ImageNet) is limited due to the difficulty of balancing toleration improvement and accuracy drop. Note the costly training or retraining will usually need to be conducted in cloud while edge devices holding quantized models focus on inference only due to the resource constraint.

Given that none of existing preventive solutions could well defend BFA after model deployment, we believe that developing a low-cost framework to detect malicious BFA in real-time with a minimized impact on the normal inference task, has become a necessity. We identify the following challenges to achieve this goal in this paper: **First**, directly detecting BFA based on the reported accuracy would bring high overhead. This is because statistically characterizing DNN's accuracy status would normally call for a large number of test patterns, e.g. at least 1000 image inferences for ImageNet with 1000 classes, leading to prominent time overhead and service interruption for just confirming BFA. **Second**, while BFA only targets flipping a few bits, there exist many different such bit combinations across various bit positions, weights and layers of an DNN model for achieving a successful attack. Moreover, not all bit-flips are as malicious as that of BFA to the models, e.g. the weight bits stored in a memory cell could suffer from random soft errors that may not degrade the accuracy during the run time. These two factors make many naive detection solutions, such as store a part of weights copy in memory and check the hamming distance (or the number of flipped bits) between current weights and the copy to correctly detect BFA in real-time, almost infeasible.

ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415726

To overcome these challenges, we propose the following key components in our framework: **First**, to make the problem amendable, we narrow the detection scope of weights to a subset–sensitive weights only, by performing a layer-wise weight sensitivity analysis based on the gradient information. The key insight is that while bit-flips in BFA could span many different bit positions, weights and layers of an DNN, statistically there exists spatial locality of such bit-flips. For example, the weight bits in certain layers may receive a higher chance of BFA than the others, though the number and position of allocated bit flips in such layers could vary. **Second**, to differentiate *"malicious" bit-flips* (BFA with desired high accuracy loss) from *"benign" bit-flips* (aka "random" bit-flips with marginal accuracy reduction in the paper), we propose to generate a detection secret-key via a light-weighted single-layer neural network, to encode those sensitive weight changes into low-dimensional binary detection code. While bit-flips in sensitive weights, regardless of benign (random) or malicious (BFA), could incur many different cases and thus are difficult to tell out (BFA or not), a proper hamming distance measurement between expected and actual low dimensional binary detection code could be a good indicator. This can be achieved by training the secret-key towards different codes with carefully produced bit-flipping variants of sensitive weights. **Finally**, BFA detection can be routinely conducted by performing secret-key based sensitive weights computation in edge devices. Once BFA is detected, repair solutions like model retraining or (reloading if a golden copy is available) will be conducted in the cloud and then an updated model will be redeployed to edge devices. Note that the cloud will handle the first two steps, while edge devices only invoke the detection routinely with the result available immediately. A detailed framework consisting of cloud and edge for BFA detection and model recovery is presented in Fig. 1.

Extensive experimental results show that our method can accurately distinguish malicious fault models under BFA from those with random bit-flips for various quantized DNN models on CIFAR-10 and ImageNet datasets, with very limited storage overhead and computation time. *To the best of our knowledge, this is the first work to address the concurrent detection problem of bit-flipping attacks against quantized DNNs popular in edge devices.*

2 Background

2.1 Quantized DNN and Notation

We consider that DNN parameters are quantized by a N_q-bits uniform quantizier, which means that the parameters with a floating-point representation are uniformly quantized into $2^{N_q}-1$ level. In hardware systems, DNN weights are normally stored as signed integer in two's complement representation. In this work, we use w and B to denote the floating-point representation and its two'2 complement, respectively. For a group of weights, the notations are $\mathbf{w}$ and $\mathbf{B}$, respectively. The binary representation of B is denoted as $\mathbf{b} = [b_{N_q-1}, \ldots, b_0]$).

2.2 Bit-Flip Attack

Bit-Flip Attack (BFA) is a gradient-based attack method, by integrating the classical fast gradient sign method (FGSM) from adversarial attack into the fault injection of quantized DNN model. FGSM adds perturbations to input $\mathbf{x}$ along its ascending gradient direction w.r.t the loss of DNN (i.e., $+\text{sign}(\nabla_{\mathbf{x}}\mathcal{L})$), while BFA flips the weight bits along the ascending gradient direction (w.r.t the loss of DNN, i.e., $+\text{sign}(\nabla_{\mathbf{b}}\mathcal{L})$). For quantized DNN model with N_q bit-width, $\nabla_{\mathbf{b}}\mathcal{L}$ can be represented as:

$$\nabla_{\mathbf{b}}\,\mathcal{L} = [\frac{\partial\mathcal{L}}{\partial b_{N_q-1}}, \ldots, \frac{\partial\mathcal{L}}{\partial b_0}] \tag{1}$$

Since $b \in \{0,1\}^{N_q}$, flipping the bit using $\text{sign}(\nabla_{\mathbf{b}}\mathcal{L}) \in \{-1,1\}^{N_q}$ could incur data overflow. As a result, BFA can be represented as:

$$\mathbf{b}^* = \mathbf{b} \oplus \left(\mathbf{b} \oplus \left(\frac{sign(\nabla_{\mathbf{b}}\mathcal{L})+1}{2}\right)\right) \tag{2}$$

where $\oplus$ is an xor operator. The attacker only performs BFA in most vulnerable bits to achieve the minimum number of bit-flips. Accordingly, a progressive bit search (PBS) is proposed to solve the following optimization problem [17]:

$$\max_{\mathbf{B}_l^*} \mathcal{L}\left(f(\mathbf{x},\mathbf{B}^*), f(\mathbf{x},\mathbf{B})\right) s.t. \sum_{l=1}^{L} \mathcal{D}(\mathbf{B}_l^*, \mathbf{B}_l) \in \{0,1,\ldots,N_H\} \tag{3}$$

where f(·, ·) is DNN function and $\mathbf{B}_l^*$ is a set of perturbed weight (2' complement) in l-th layer (total number of layer is L). $\mathcal{D}(\mathbf{B}_l^*, \mathbf{B}_l)$ represents the hamming distance between the clean weight and bit-flipped weight, and N_H is the maximum Hamming distance allowed. The goal of PBS algorithm is to identify the most vulnerable bits from $\mathbf{B}_l$ through the gradient ranking, so as to conduct the BFA with Eq. 2. Then the algorithm identifies the j-th layer with maximum loss (i.e., j=arg $\max_l\{\mathcal{L}_1, \ldots, \mathcal{L}_l, \ldots \mathcal{L}_L\}$) and re-performs BFA. In each iteration, it will perform BFA in the most vulnerable bit of one layer until BPA brings significant accuracy degradation or reaches $\mathcal{D}(\mathbf{B}_l^*, \mathbf{B}_l) > N_H$.

2.3 Threat Model

We adopt a similar threat model from [17]. The parameters of a DNN model, including the weights, bias and batch-normalization parameters, are stored in DRAM and loaded in cache before performing computation. We assume that the attacker is capable of accessing model structure, model parameters and partial training datasets. The attacker can utilize gradient-based BFA to search vulnerable bits and perform the Row-hammer attack to achieve bit-flipping in DRAM. The defender has access to the weights and secret-key stored in edge devices for performing BFA detection.

3 Overview

Fig. 1 depicts an overview of the proposed defense framework consisting of two key modules: malicious BFA detection and model recovery. We assume that during model deployment, a DNN provider trained a quantized DNN model and deployed it into the DRAM of the edge device. The Bit-Flip Attack can inject malicious bit-flips to weights stored in DRAM, which will incur the extreme accuracy degradation of the model. Then we can introduce our detection mechanism, which includes the following key steps: **1)** weight sensitivity analysis; **2)** detection secret-key generation; and **3)** detection code generation. First, weight sensitivity analysis is performed to pick only sensitive weights as the weight detection candidates. Second, the detection secret-key generation algorithm generates the detection secret-key by encoding weight detection candidates into binary detection code. Third, the detection mechanism routinely

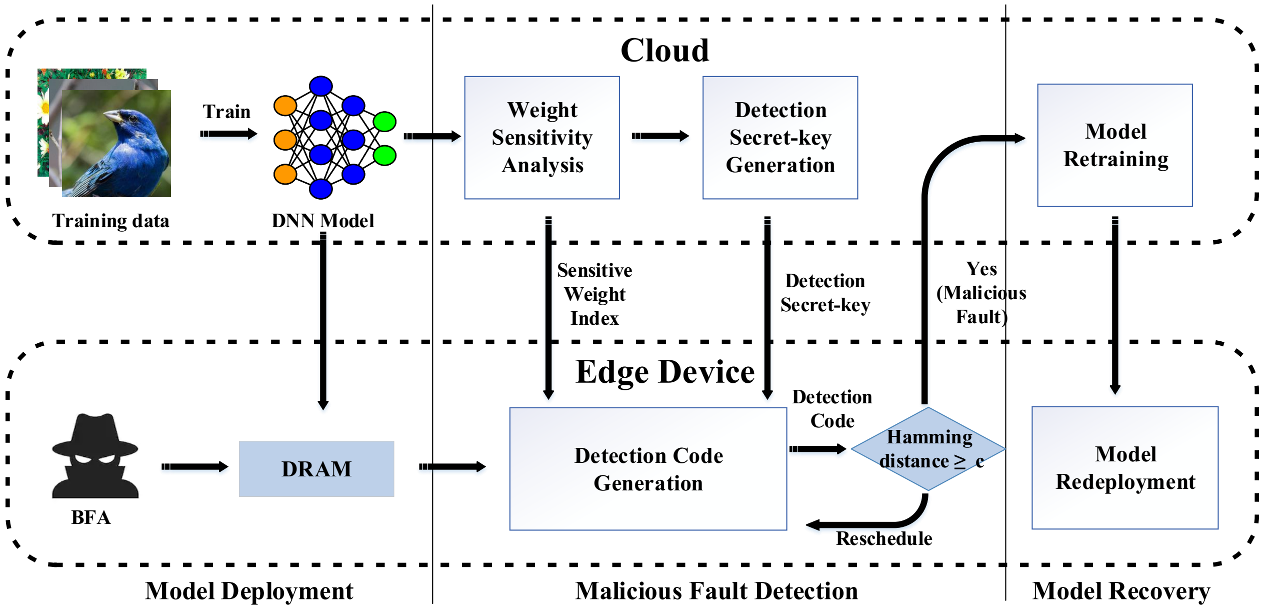

Figure 1: An overview of the defense framework with BFA detection and model recovery.

outputs detection code using detection secret-key and weight detection candidates and calculate the Hamming distance between the current detection code and original detection code. By comparing the Hamming distance with a defined threshold c, we can detect malicious BFA. If the model is under BFA, retraining the DNN model for a few epochs can quickly recover the model accuracy. Note the original retraining process, can easily fine-tune other weights to adapt to the bit changes for accuracy recovery given that BFA only flips a few weight bits. As we shall show later, with only one epoch retraining, the top-1 accuracy of ResNet-34 under BFA can be boosted from $\sim$ 1% to 73% on ImageNet dataset.

Our framework requires a collaborative effort between the edge and cloud based on the needed resource for each step. As Fig. 1 shows, the expensive weight sensitivity analysis and the detection secret-key generation will be operated in the cloud. The produced index of sensitive weights and the detection secret-key will then be stored in the edge device. Consider that the resource limitation on edge, the storage and computation overhead should be very low, e.g. storage overhead is much less than that of the quantized DNN model itself. Then edge device, which focuses on on-device inference, will utilize these information to quickly generate the detection code for BFA detection in real-time. The detection can be performed routinely with minimum impact on normal inference. The repairing process like model retraining, should be done in cloud once a BFA is detected. Finally, the retrained model will be re-deployed to the edge device.

Since our work primarily focuses on BFA detection, we will introduce the associated technique details in the next section.

4 Detection Procedure

To detect malicious BFA in real-time, we need to design a detection flow to reduce the overhead while precisely detecting the malicious bit-flip in a practical scenario from the random benign bit flipping.

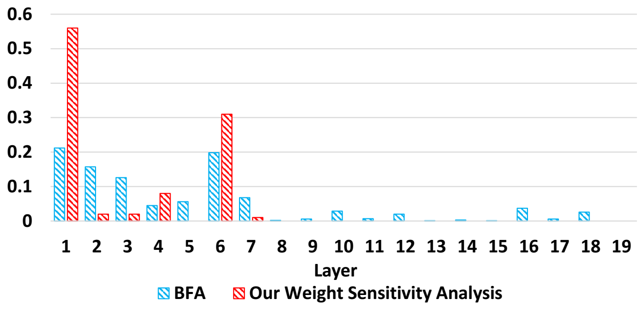

Figure 2: The normalized average bit-flip rates in actual BFA (50 trials) and those of $top_{100}|\nabla_{\mathbf{B}}\mathcal{L}|$ in our weight sensitivity analysis in each layer of ResNet-20 on CIFAR10.

In this section, we propose a novel weight encoding-based detection (WED) mechanism to solve the above problem.

4.1 Weight Sensitivity Analysis

Since state-of-the-art DNN models contain million-to-billion weight parameters, so our first step is to reduce detection scope by picking those sensitive weights that are more likely to receive the malicious bit-flips. If the malicious bit-flip occurs in those weight detection candidates, the model has a high probability of BFA. We perform the weight sensitivity analysis to select a fraction of weights from all weights as the weight detection candidates. Considering that BFA is a gradient-based attack method by flipping the bit with the largest $|\frac{\partial \mathcal{L}}{\partial b}|$, we adopt the gradient information to measure the weight sensitivity. Here we refer to the gradient of weight B instead of the gradient of bit (note, $\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial B}\frac{\partial B}{\partial b}$). The weights whose absolute values of the gradients $|\nabla_{\mathbf{B}}\mathcal{L}|$ are top-n are selected as the weight detection candidates. However, since those weights could be located in any layer for an DNN model, it is too costly to cover

Algorithm 1: Detection Secret-key Generation

```
    Data:
    $\mathbf{B} \in \mathbb{Z}^N$; // weight detection candidates
1   $T$; // the batch size
2   $\mathbf{t} \in \{0,1\}^{T \times M}$ // the label of detection code, e.g. each t is $[1,1,\dots,1]^M$
3   $c$; // the tolerance coefficient
4   $p$; // the possibility for bit-flip randomly
5   $\alpha, epoch$; // learning rate and the maximal iterations
    Result:
    $K$; // detection secret-key
6   $K \leftarrow$ initialize K with random noise, i.e., $K \in \mathcal{N}(0,1)$
    // define the perceptron function and the loss function for batch
7   $\mathbf{y}(\mathbf{W},\mathbf{K}) = \phi(\mathbf{W} \times \mathbf{K})$
8   $\mathcal{L}_d(\mathbf{y},\mathbf{t}) = -\frac{1}{T}\sum_{j=1}^{T}(\mathbf{t}\log\mathbf{y}(\mathbf{W},\mathbf{K}) + (\mathbf{1}-\mathbf{t})log(\mathbf{1}-\mathbf{y}(\mathbf{W},\mathbf{K})))$
9   while $i < epoch$ do
10      // fabricate training and testing data
11      while $j < T$ do
12          $\mathbf{B}_b^{train}, \mathbf{B}_b^{test} \leftarrow$ randomly flip $b_{N_q/2} - b_0$ with possibility $p$
13          $\mathbf{B}_m^{test} \leftarrow$ randomly flip $b_{N_q}$ with possibility $p$
14          j = j + 1
15      // preprocessing of training data
16      $\mathbf{W}_b^{train} \leftarrow$ convert two's complement to floating-point and perform shuffle
17      $g = \nabla_{\mathbf{K}}\mathcal{L}_d(\mathbf{y}(\mathbf{W}_b^{train},\mathbf{K}),\mathbf{t})$
18      $\mathbf{K} = \mathbf{K} - \alpha * g$
19      if $\mathcal{D}(r(y(B_b^{test},K)),t) == 0$ and $\mathcal{D}(r(y(B_m^{test},K)),t) \geq c$ then
20          break;
21      i = i + 1
22  return K
```

all the layer when extracting them from non-consecutive memories by index for the detection code generation. Instead, we found that the majority of the top-n gradients could be allocated in one or two layers. And statistically the same layers should also have the most number of bit flips in BFAs. Note that the bit-flip combination for a successful attack could vary from one BFA to another against the same model.

Fig 2 shows the normalized bit-flip distribution in each layer when performing BFA for 50 trials in ResNet-20 on CIFAR10, as well as the top-100 gradient distribution characterized by our weight sensitivity analysis. For actual BFAs, we perform the attack algorithm 20 iterations to flip 20 bits. As Fig.2 shows, most bit-flips occur in layers one and six (the top two blue bars), which is consistent with the layers identified by our weight sensitivity analysis (the top two red bars). This confirms that bit-flipping of BFA preserves a layer-wise spatial locality property, i.e. the neighborhood of the sensitive weights with bit-flipping of BFA is likely to have the malicious bit-flips of BFA again. As a result, we calculate the proportion of $top_n| \nabla_{\mathbf{B}} \mathcal{L}|$ in each layer and select the weights in a few layers that have a large proportion (i.e., top_1 or top_2) as weight detection candidates. This process will significantly reduce the overhead caused by indexing weight detection candidates.

4.2 Weight Encoding Detection (WED)

The weight encoding attempts to transform weight detection candidates into low-dimensional binary detection code **d**. Given weight detection candidates $\mathbf{B} \in \mathbb{Z}^N$ and detection secret-key $\mathbf{K} \in \mathbb{R}^{N \times M}$ (M is the bit-width of detection code), the transformation process can be formulated as:

$$d_j = r(y_j),\ y_j = \phi(\sum_{i=0}^{N-1} B_i \cdot K_{ij}) \tag{4}$$

where r(·) is the the round function and $\phi(\cdot)$ is the sigmoid function. It can be viewed as a single-layer perceptron with the detection secret-key as its parameters. The goal of the perceptron is to generate a transform matrix **K** (or secret-key) to encode weight detection candidates to detection code by learning malicious and benign pattern in weight. If weights have malicious fault injection, the generated detection code will be different from the original detection code of clean model. We can utilize the Hamming distance to measure this difference. A larger Hamming distance indicates a high probability of malicious BFA. For random or benign bit-flippings, the generated detection code will be same as the original detection code of clean model.

We use binary cross entropy as our loss function:

$$\mathcal{L}_d = -\sum_{j=1}^{M}\left(t_j \log y_j + (1-t_j)log(1-y_j)\right) \tag{5}$$

where t is the label of detection code ($\mathbf{t} \in \{0,1\}^M$). For simplicity, we set **t** as M bits all one code in our experiment. In the training stage, we generate the training data by weights with benign bit-flippings. Considering that the malicious bit-flipping is more likely to happen in higher bits, while benign bit-flipping is not, to simulate benign bit-flip, we randomly flip $[b_{N_q/2},\dots,b_0]$ with a probability p to produce the training data $\mathbf{B}_b^{train}$. Generally p is a very small value (e.g. 1%) with almost no model accuracy reduction. Then we utilize gradient descent method to solve this optimization problem–$\min_{\mathbf{K}} \mathcal{L}_d$. In the testing stage, we need to calculate the Hamming distance between detection code **d** of benign testing data $\mathbf{B}_b^{test}$ and label to verify if it is zero. On the other hand, we also need to test the malicious BFA. Note the training process does not involve any malicious bit-flipping injection because it is a two-class problem, that is, if the result satisfies the minimum Hamming distance, the model is benign; otherwise, the model could have the malicious fault. Similar to simulating benign bit-flip, we consider flipping higher bits as the malicious bit-flipping because it has a higher possibility to cause lager weight shift to degrade accuracy. Thus in our algorithm, we randomly flip "most malicious" b_{N_q} with the possibility p to mimic the malicious testing data $\mathbf{B}_m^{test}$ to test our detection secret-key. We calculate the Hamming distance between detection code of weights with malicious bit-flip and label **t** to verify if it is larger than a certain positive constant. We name this constant as tolerance coefficient c (c is a positive integer, e.g., 3). If the Hamming distance is larger than the tolerance coefficient, it means that model has malicious fault injection. The details of detection secret-key generation in cloud is showed in algorithm 1. To accelerate the model convergence, our algorithm trains perceptron with batch processing. Eq. 4 will be used to generate actual detection code for quantized models deployed on edge.

5 Evaluation

5.1 Experimental Setup

Datasets. We evaluate our detection framework on two datasets: CIFAR-10 [13] and ImageNet [14]. CIFAR-10 is a 32×32 RGB dataset with 10 classes and consists of 60k RGB images (50k for training and 10k for testing). ImageNet is a more complex image classification dataset with 1000 classes and 1.2M images.

DNN Model and Quantization Setting. We adopt ResNet-20[6] with 8-bit weight quantization to classify the CIFAR-10 dataset, and its quantized accuracy is 89.17%. For ImageNet, the two 8-bit quantized deep models–ResNet-34 and MobileNet [10] are used. The original top-1 accuracies of the well-trained quantized ResNet-34 and MobileNet are 73.13% and 68.51%, respectively.

Fault Models. Two kinds of bit-flipping fault models are considered in our experiments. The first one is the malicious fault model under BFA, which flips a few of the most vulnerable bits to achieve a destructive accuracy degradation for quantized DNN. We conduct BFA with 20 iterations to flip 20 bits in each quantized network candidate. Under BFA, their top-1 accuracies are dropped to 10.33% (ResNet-20), 0.1% (ResNet-34) and 0.1% (MobileNet), respectively, which are close to a random guess. We perform this configuration with 50 different random seeds to construct 50 malicious fault models for each network candidate. The second one is the benign fault model with random soft error (i.e., random bit-flips) in run-time. We inject random bit-flip fault to ResNet-20, ResNet-34 and MobileNet with the p as 0.25%, 0.1% and 0.1%, respectively (i.e., flipping 670, 2177 and 347 bits) to achieve slight accuracy degradation (i.e., less than 2%). We repeat this process 50 times to construct 50 benign fault models for each model candidate. In total we have 100 fault models for each network candidate.

Detection Configuration. For weight sensitivity analysis, we randomly draw a batch of images (128 for CIFAR-10 and 256 for ImageNet) from the test data to calculate the proportion of $top_{100}|\nabla_{\mathbf{B}}\mathcal{L}|$ in each layer and select the weights in sensitive layers with the top_1 or top_2 percent as the weight detection candidates. We represent those two detection settings as WED(1) or WED(2) in our evaluation. Table 1 shows the sensitive layer index (which layer) and the number of weights in the corresponding layer. For example, for ResNet-20, top_1 sensitive layer is the first layer, so we select 432 weights in the first layer as weight detection candidate of WED(1). While the top_2 sensitive layers include both the first and 6th layer, leading to a total of 2736 weights (432+2304) in WED(2). Note that the total number of weights are 267696 in ResNet-20, so those in WED(1) and WED(2) only represent a small portion of that. For the detection secret-key generation, we adopt the following settings: a learning rate $\alpha = 0.1$, $epoch = 500$, batch size $T = 64$ and the tolerance coefficient $c = 3$ in algorithm 1. We select 32 and 64 bit widths for detection code in CIFAR-10 and ImageNet, respectively. All "1" binary code is selected as the original detection code To further reduce overhead, we quantize detection secret-key to 4 bits.

Metrics. We use 6 metrics in our evaluation. We first define how to decide whether the model is under BFA or not in our detection mechanism: if the Hamming distance of detection codes between the fault model and the clean model is larger than a threshold c, the target model is under BFA (malicious fault model); otherwise, it is under non-BFA (or benign fault model). Our first metric is the True Positive Rate (TPR):

$$TPR = \frac{TP}{TP+FN} = \frac{malicious\ fault\ models\ correctly\ identified\ as\ malicious}{the\ number\ of\ malicious\ fault\ models} \tag{6}$$

Since we select two thresholds $c = 1$ and 3, the corresponding metrics are denoted as TPR-1 and TPR-3, respectively. The second metric is the True Negative Rate (TNR):

$$TNR = \frac{TN}{TN+FP} = \frac{benign\ fault\ models\ correctly\ identified\ as\ benign}{the\ number\ of\ benign\ fault\ models} \tag{7}$$

Similarly we have TNR-1 and TNR-3 for $c = 1$ and 3. The third metric is Detection Rate (DR), which can be expressed as:

$$DR = \frac{TP+TN}{TP+FN+TN+FP} \tag{8}$$

We then also have DR-1 and DR-3 for $c = 1$ and 3. As a result, TPR-1, TPR-3, TNR-1, TNR-3, DR-1 and DR-3 will be used to comprehensively evaluate our detection approach.

5.2 Results and Analysis

Figure 3 reports the Hamming distance of detection codes and the number of bit-flips in our weight detection candidate–WED(1) for various malicious fault models under BFAs (ResNet-20 on CIFAR-10, ResNet-34/MobileNet on ImageNet). For simplicity, we show the first 10 samples of 50 malicious fault models of each network candidate as an example in Fig. 3. First, we observe that many malicious bit-flips occur in our weight detection candidates. As Figure 3(a) shows, each malicious fault model exhibits a considerable number of malicious bit-flips (the average number is 5/20) in weight detection candidate of ResNet-20. Note our weight detection candidates only represent a small portion of all weights in ResNet-20 (432 v.s. 267696). This result demonstrates that our weight sensitivity analysis can correctly pick sensitive weights that are more likely to be injected with malicious bit-flips by BFA. We observe the similar result in ResNet-34 on ImageNet (see Figure 3(b)). However, as Figure 3(c) reports, the number of malicious bit-flips in MobileNet can be significantly less than that in ResNet-20 and ResNet-34. This is because selecting only one sensitive layer from the deeper MobileNet (53 layers v.s. 20/34 in ReseNet) to cover many malicious bit-flips is difficult. Second, we observe that the hamming distance between the detection code in malicious fault models and its original version is quite large. For example, in Figure 3(a) and 3(b), the Hamming distance is more than 1 or 3 (the threshold c) in most fault models, which means that most of the malicious fault models could correctly be identified as BFA by our WED(1) in ResNet-20 and ResNet-34. For MobileNet, the Hamming distance in many fault

Table 1: The detailed configurations of Weight detection candidates in weight sensitivity analysis

Dataset	Structure	top_2 sensitive layers	#weights in each sensitive layer	total number of weights
CIFAR-10	ResNet-20	[1,6]	[432,2304]	267696
ImageNet	ResNet-34	[10,1]	[8192,9408]	21779648
	MobileNet	[2.3]	[288,512]	3469760

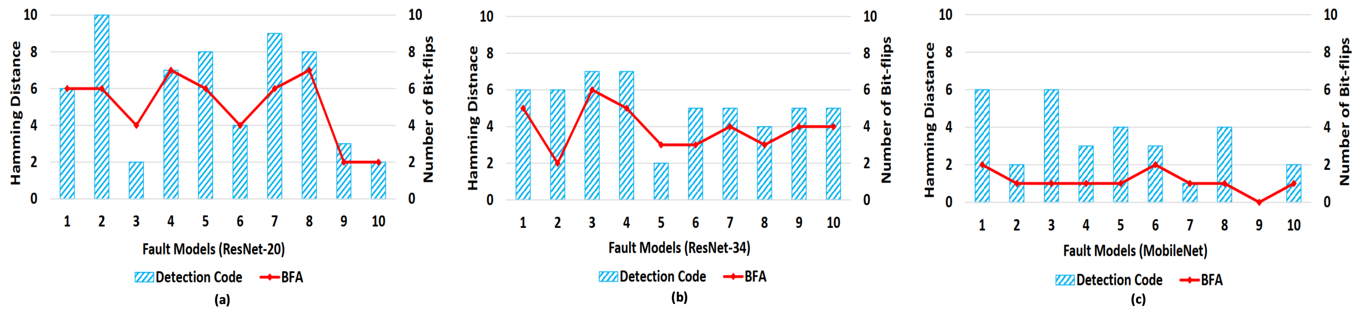

Figure 3: The Hamming distance between original and tampered detection codes, and the number of bit-flips in our weight detection candidates–WED(1) in malicious fault models under BFA in ResNet-20, ResNet-34, MobileNet for CIFAR-10 and ImageNet.

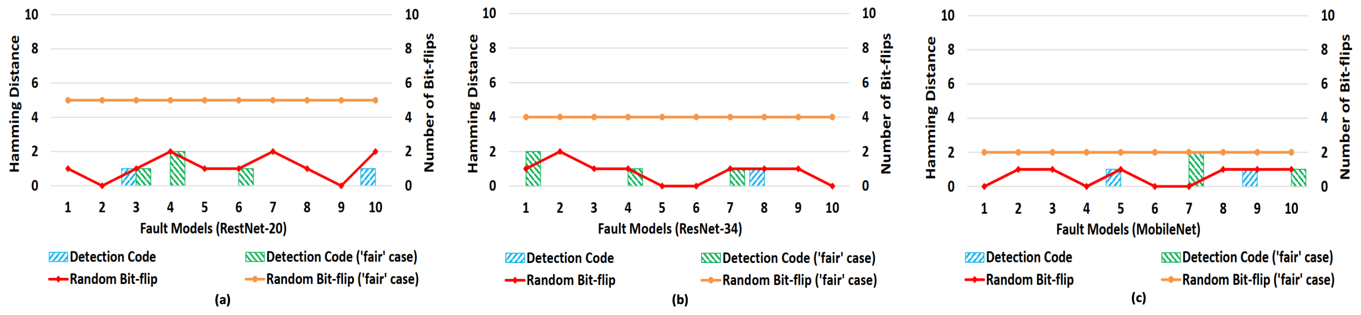

Figure 4: The Hamming distance of between original and tampered detection codes, and the number of bit-flips in our weight detection candidates–WED(1) in benign fault models with random bit-flipping in ResNet-20, ResNet-34, MobileNet for CIFAR-10 and ImageNet.

models is smaller than that in ResNet-20 and ResNet-34 (see Figure 3(a)). The Hamming distance could become even as low as 0 in fault model 9, because there are no malicious bit-flips in weight detection candidates. Third, a larger number of malicious bit-flips could lead to more considerable Hamming distance. This indicates that if more malicious bit-flips occur in our weight detection candidates, the accuracy of detection could be higher. Therefore, we choose WED(2) which includes more weight detection candidates, to improve the detection accuracy for deeper networks like MobileNet.

For benign fault models with random bit-flipping, as shown in Figure 4, the Hamming distances between detection codes in clean- and most benign fault models are 0 across various network candidates for both CIFAR-10 and ImageNet. For a comprehensive comparison with malicious fault model under BFA, we add an additional benign fault model as a 'fair' case here by injecting more bit-flips, e.g. the number equals to the average of that in the malicious model (5 bit-flips for ResNet-20), into our weight detection candidates (see the green bar and orange line in Fig. 4).

For the normal case (the configuration in 5.1), the limited benign bit-flips seem to rarely occur in our weight detection candidates when compared with malicious bit-flips. The reason is because our weight detection candidates are only a small portion of all weights, therefore the benign bit-flippings which could occur across the entire model randomly will have a lower probability to flip bits in our weight detection candidates. Even for the 'fair' case which could have much more random bit-flippings appear in our weight detection candidates, we still observe that the Hamming distance in most benign fault models is 0. This means that our detection method correctly distinguishes BFA and random bit-flippings even under such a setting. Note although the random bit-flipping does not have the spatial locality property, the numerous random bit-flips in our weight detection candidates mean that the model could have suffered from a significant accuracy degradation. Considering this kind of random bit-flipping is already not benign, we do not consider such case in our evaluation. Beside, we also found that there exists non-zero Hamming distance in a few benign fault models, as shown in Fig. 4. To address this issue, we can set a threshold $c > 1$ (e.g. 3) instead of 0 in practical detection. The reason is because our detection secret-key generation algorithm 1 assumes that benign bit-flips and malicious bit-flips occur in the $[b_{N_q/2}, \ldots, b_0]$ and b_{N_q}, respectively, while that of random bit-flips could appear in any bit of a weight in practice.

Bit-width Selection for Detection Code. The length of the detection code is a critical factor that impacts the detection effectiveness in our detection approach. Table 2 reports the average Hamming distance between original detection codes (no bit-flips) and fault models (malicious/ random bit-flips) with different code length selections for WED (1). We select 50 benign and 50 malcious fault models in this evaluation. For malicious bit-flips, as the code length increases, the average Hamming distances will increase significantly. For example, if the code length increases from 16 to 128 bits, the Hamming distance could increase from 3.86 to 11.94 in ResNet-20. A larger average Hamming distance in malicious fault models indicates a better detection effectiveness, e.g. better TPR, because of a higher possibility of exceeding a preset threshold. However, for the benign fault model with random bit-flips, a larger

Table 2: The average Hamming distance between original (clean models) and tampered (fault models) detection codes with different bit-widths for WED(1).

Dataset	Structure	Fault Model	Bit-width of detection code			
			16-bits	32-bits	64-bits	128-bits
CIFAR-10	ResNet-20	Malicious Bit-flips (BFA)	3.86	4.32	6.64	11.94
		Random Bit-flip	0.42	0.44	0.56	0.84
ImageNet	ResNet-34	Malicious Bit-flips (BFA)	3	3.34	4.92	9.22
		Random Bit-flip	0.16	0.21	0.42	0.82
	MobileNet	Malicious Bit-flips (BFA)	1.02	2.36	3.08	9.87
		Random Bit-flip	0.14	0.19	0.49	1.14

Table 3: The detection effectiveness of our WED(1) and WED(2) in various detection metrics in ResNet-20, ResNet-34, MobileNet for CIFAR-10 and ImageNet

CIFAR-10							
Structure	Detection Setting	TPR-1	TPR-3	TNR-1	TNR-3	DR-1	DR-3
ResNet-20	WED(1)	90%	80%	88%	100%	89%	90%
	WED(2)	96%	90%	88%	100%	92%	95%
ImageNet							
ResNet-34	WED(1)	100%	82%	96%	100%	98%	91%
	WED(2)	100%	94%	96%	100%	98%	97%
MobileNet	WED(1)	80%	58%	88%	100%	84%	79%
	WED(2)	100%	98%	94%	100%	97%	99%

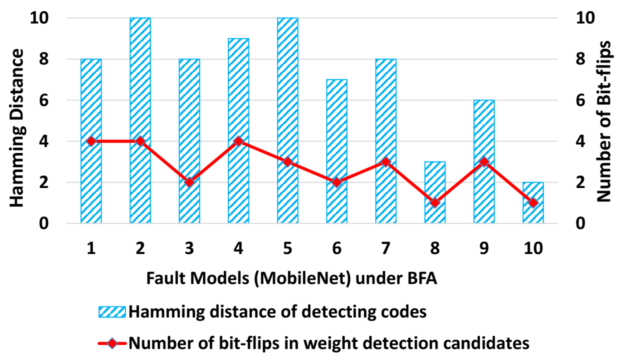

Figure 5: The Hamming distance between original (clean model) and tampered (fault models) detection codes, and the number of bit-flips in 10 malicious fault models under BFA in MobileNet on ImageNet for our WED(2).

code length only leads to a slight increase of Hamming distance. For example, in ResNet-20, the Hamming distance increases from 0.42 to 0.84, when increasing the code length from 16 to 128 bits. On the other hand, a too large Hamming distance can also hurt the detection effectiveness, e.g. a lower TNR. To correctly identify a benign fault model as benign, the Hamming distance should be less than the threshold. In MobileNet, the average Hamming distance of detection code with 128 bits can even reach up to 1.14, which exceeds the threshold 1. Besides, a large code length would bring high overhead in detection. Therefore, the length of the detection code should be moderate, in order to achieve better detection effectiveness and low overhead. We select 32 for ResNet-20/ResNet-34 and 64 for MobileNet in our evaluation.

Detection Effectiveness. Table 3 shows the detection effectiveness of our WED(1) and WED(2) in various detection metrics in three network candidates on CIFAR-10 and ImageNet. Our WED(1) achieves competitive detection effectiveness in various detection metrics in ResNet-20 and ResNet-34 (both CIFAR-10 and ImageNet). A high TPR or TNR means that our detection approach can correctly detect malicious or benign fault models. For MobileNet, we observe our WED(1) achieves a lower TPR-3, i.e., 58%, which is consistent with our analysis in Figure 3(c). For a deeper neural network structure (e.g., > 50), more layers should be selected as the weight detection candidates. As Fig. 5 shows, more malicious bit-flips are observed in WED(2) and could bring a more considerable Hamming distance. As a result, WED(2) achieves 100% and 98% in TPR-1 and TPR-3 in MobileNet, respectively (see Table 3). Similarly, WED(2) can also improve detection effectiveness for ResNet-20 and ResNet-34. For example, our WED(2) achieves 92% (98%) and 95% (97%) in ResNet-20 (ResNet-34) on CIFAR-10 (ImageNet) in DR-1 and DR-3, respectively.

Detection Overhead. For the detection code generation, we extract weight detection candidates by index and calculate its detection code using Eq.4. Therefore, the overhead mainly comes from the matrix multiplication and weight extraction. Since our weight detection candidates with a layer-wise design could incur continuous memory accesses, the weight extraction cost is expected to be low. The overhead of matrix multiplication depends on the number of weights in the detection candidates, which could be low as well given that only a small portion of model weights selected in our detection. Table 4 reports both the timing and storage overhead of our detection approach in ResNet-20, ResNet-34 and MobileNet on CIFAR-10 and ImageNet. First, our detection approach achieves a very low time overhead in various network candidates, despite that more weights could increase overhead. For example, the average time cost of our WED(1) in 100 fault ResNet-20 models is 0.00059s, while that of our WED(2) is 0.00092s. Second, the storage cost in our detection approach is related to the number of selected weights, bit width of detection secret-key, and the length of detection code. For example, in ResNet-20, the storage cost of our WED(1) can be estimated as $432 \times 4 \times 32 + 432 \times 16$ bits (here we assume that storing the index uses 16 bits integer). As Table 4 shows, all WED(1) and most WED(2) require very low storage cost across various network

Table 4: The timie and storage overhead of our detection approach in ResNet-20, ResNet-34 and MobileNet for CIFAR-10 and ImageNet.

Structure	Detection	Time Cost (s)	Storage Cost (MB)
ResNet-20	WED(1)	0.00059	0.007 (2.7%)
	WED(2)	0.00092	0.047 (18.4%)
ResNet-34	WED(1)	0.0013	0.141 (0.68%)
	WED(2)	0.0022	0.302 (1.45%)
MobileNet	WED(1)	0.0009	0.009 (0.27%)
	WED(2)	0.0012	0.026 (0.78%)

Table 5: The accuracies of original model, fault model under BFA and recovered model in ResNet-20, ResNet-34 and MobileNet for CIFAR-10 and ImageNet.

Structure	Original Model	Model under BFA	Recovered Model
ResNet-20	89.17%	10.88%	88.58%
ResNet-34	73.13%	0.1%	72.41%
MobileNet	68.51%	0.1%	67.68%

candidates. For example, the storage cost of our WED(1) in ResNet-34 is only 0.141 MB (0.68% of the entire model). Only WED(2) in ResNet-20 incurs a relatively high storage cost, because of selecting 2-layer weights out of the 20 layers. However, since the detection accuracy of WED(2) with much increased overhead in ResNet-20 is only $< 5\%$ higher than WED(1), while WED(1) can deliver $\sim 90\%$ in DR-1 and DR-3, we may use WED(1) for a good balance between detection effectiveness and overhead. However, in general, WED(1) is more suitable for a simple model (e.g., the number of layers $\leq$ 20), while WED(2) can achieve outstanding detection effectiveness and low overhead in the complicated (deeper) model.

Model Recovery. Table 5 shows the accuracies of the original model, the fault model under BFA and the recovered model. Compared with the original accuracy, the recovered accuracy only has a slight degradation (i.e., $< 1\%$) in various network candidates on both CIFAR-10 and ImageNet. Specifically, for CIFAR-10 (ResNet-20), we retrain fault model under BFA for just 1 epoch using the original training algorithm, the accuracy can be recovered from 10.88% to 88.58%. For ImageNet (ResNet-34 and MobileNet), to accelerate our retraining process, we randomly draw 10% images from the training dataset, and retrain the fault model in one epoch. Again, we can quickly recover the fault model's accuracy to the original level, although BFA leads to extreme accuracy degradation. The underlying reason is because BFA only flips a few of bits of a few weights, a simple retraining process could easily fine-tune other weights to adapt such bit change.

6 Conclusion

In this work, we propose a low-cost detection approach to correctly and quickly detect the emerging Bit-Flip Attack (BFA) against quantized DNNs popular in edge devices. To achieve a low overhead, we first perform a layer-wise weight sensitivity analysis to select only a small portion–most sensitive weights as weight detection candidates. To distinguish the malicious bit-flips under BFA and benign bit-flips brought by the common soft errors, we design a gradient descent algorithm to optimize loss function to embed the knowledge about two kinds of bit-flips into the detection secret-key. The sensitive weight changes will be encoded into the binary detection codes through the detection secret-key based transformation that responses to malicious and benign bit-flips differently. By computing the Hamming distance between the detection code generated by the current weight and original detection code, we can detect BFA in a real-time manner with minimized impact on normal inference on edge devices. As a result, our detection approach achieves high BFA detection effectiveness with limited overhead across various quantized DNN models.

7 Acknowledgements

This work was supported in part by NSF Grants CNS-2011260 and CCF-2011236.

References

[1] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. IEEE, 39–57.

[2] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 55–71.

[3] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[4] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. 2018. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 245–261.

[5] Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Semeen Rehman, and Muhammad Shafique. 2018. Robust machine learning systems: Reliability and security for deep neural networks. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 257–260.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[7] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. [n.d.]. Defending and Harnessing the Bit-Flip based Adversarial Weight Attack. ([n. d.]).

[8] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraş. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 497–514.

[9] Xiaolu Hou, Jakub Breier, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2019. Experimental Evaluation of Deep Neural Network Resistance Against Fault Injection Attacks. *IACR Cryptology ePrint Archive* 2019 (2019), 461.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[11] Yan Xiong Liangliang Chang Zhezhi He Deliang Fan Jingtao Li, Adnan Siraj Rakin and Chaitali Chakrabarti. 2020. Defending Bit-Flip Attack through DNN Weight Reconstruction. In *57th Design Automation Conference (DAC)*. IEEE.

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

[13] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2010. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html* 8 (2010).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning attack on neural networks. (2017).

[16] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. 2017. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 131–138.

[17] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search. In *Proceedings of the IEEE International Conference on Computer Vision*. 1211–1220.

[18] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15 (2015), 71.