Safeguarding the Intelligence of Neural Networks with Built-in Light-weight Integrity MArks (LIMA)

Fateme S. Hosseini*, Qi Liu†, Fanruo Meng*, Chengmo Yang*, Wujie Wen†
*University of Delaware
{fateme, fanruo, chengmo}@udel.edu
†Lehigh University
{qil129, wuw219}@lehigh.edu

Abstract—**As Deep Neural Networks (DNNs) are widely adopted in many real-world applications, their integrity becomes critical. Unfortunately, DNN models are not resilient to fault injection attacks. In particular, recent work has shown that Bit-Flip Attack (BFA) can completely destroy the intelligence of DNNs with a few carefully injected bit-flips. To defend against this threat, we propose Light-weight Integrity MArks (LIMA) framework which protects the integrity of the most significant bits (MSBs) of DNN weights – the main target of BFA. Such protection is enabled by embedding specific property into a trained DNN model's weights before deploying it in hardware. LIMA outperforms existing BFA countermeasures as it requires no retraining, imposes no storage overhead, offers full-coverage of all DNN layers, and can be easily verified with Multiply-Accumulate (MAC) operations to detect BFA. Our comprehensive study demonstrates 100% effectiveness in detecting chains of bit-flips and near-zero accuracy loss for embedding LIMA. The results also show that even when the attacker has complete knowledge of the proposed defense plan, attacking DNNs with built-in LIMA is extremely difficult, if not completely impossible.**

I. Introduction

With tremendous success in solving a variety of machine learning tasks, Deep Neural Networks (DNNs) have gained ever-increasing popularity across many application domains. Meanwhile, their integrity has turned into a major concern, especially when deployed in mission-critical applications such as autonomous driving and medical diagnosis [1]–[4]. Unfortunately, DNNs are not resilient to faults in the inputs or in the network structure. Both *adversarial examples* (i.e., maliciously crafted inputs) [5]–[10] and *fault injection attacks* (i.e., maliciously altered DNN parameters) [11]–[13] can lead to severe accuracy drop.

Recently, a new type of fault injection attacks called Bit-Flip Attacks (BFAs) [14] dedicated to hardware-friendly quantized DNN models has been developed. BFAs can crash DNN models deployed in hardware to near-random guessing by injecting a chain of precisely selected bit-flips into model parameters, e.g., by flipping only 13 bits out of million parameters of an 8-bit quantized Resnet-18 model, its accuracy drops from 69.8% to 0.1% [14]. A follow-up work demonstrated the feasibility of launching BFAs in Intel i7-3770 CPU with DDR3 memory via a new type of precise rowhammer attacks designed for BFA [15]. Most recently, a stealthy and probably more dangerous variation of BFAs called *Targeted Bit-Flip Attack* (T-BFA) [16] was proposed, which can go unnoticed for a long time as it only alters the classification of some inputs and only causes a slight drop in the model's overall accuracy. These works turn BFAs into a real and serious security threat, especially to machine-learning-as-a-service (MLaaS) platforms [17] that offer inference services to multiple users.

Among the existing countermeasures, input-based testing methods [18]–[20] and general Rowhammer mitigation techniques [21]–[28] seem to be applicable to BFAs, at first sight. The first category uses a small set of sensitive images to detect accuracy drop. However, no single image can be sensitive enough to detect bit-flips that may occur in any of the millions (or even billions) of weights in a DNN. Input-based testing is especially ineffective for detecting stealthy T-BFAs or even an early terminated BFA, as both schemes only cause small drop in overall model accuracy which would require a large set of test images to detect, as confirmed by [18], [20]. The second category, i.e., general rowhammer mitigation, is not well-suited to BFAs either. Standard Error Correcting Code (ECC) and Intel Software Guard Extensions (SGX) can be broken by the new type of Rowhammer attacks as shown in [21], [22], not to mention that these defenses require non-trivial hardware modifications such as additional/probabilistic refreshes,

This work is partially supported by Semiconductor Research Corporation grant #2964.001, National Science Foundation grants #1909854 and #2011236.

extra row write counters, and redesign of the memory controller. Recently, several BFA-specific countermeasures have been proposed [29]–[31]. However, as what will be shown in Section II, they either alter the DNN weight distribution, thus affecting inference accuracy and making the defense easily detectable by an attacker [29], [30], or are designed to protect only a subset of network layers and hence can be easily bypassed by an attacker with knowledge of the defense [31].

Given the severity of BFAs and the inability of existing approaches to fully address it, we propose **Light-weight Integrity MArks (LIMA)**, the first comprehensive defense against BFAs. LIMA is designed based on the observation that BFAs typically flip the Most Significant Bits (MSBs) of weights to create largest value deviation, while the Least Significant Bits (LSBs) remain intact. Accordingly, LIMA's integrity marks are embedded by adjusting the LSBs to establish an equality relation between MSBs and LSBs of the sum value of a group of weights, allowing for quick run-time verification. Embedding LIMA's integrity marks into a DNN causes near-zero accuracy drop and requires no extra storage or special hardware support. Verifying the integrity marks requires only Multiply-Accumulate (MAC) operations which are commonly available in DNN accelerators and software implementations. We will show, through a thorough study of multiple DNN models, that LIMA is not only capable of detecting every single chain of bit-flips chosen by BFAs and T-BFAs, but also inducing no noticeable drop to the model's original accuracy.

The rest of this paper is organized as follows: Section II introduces BFA and its variations. Section III outlines the threat model. Section IV describes the defense paradigm and the technical details for embedding the integrity marks in DNN and verifying them at runtime. Section V experimentally evaluates LIMA in terms of BFA detection, while Section VI presents an in-depth discussion on LIMA's ability to detect stealthy T-BFAs, its impact on weight distribution and robustness against bypassing, and possible recovery methods. Finally, Section VII concludes the paper.

II. Preliminaries on BFA

Fault injection attacks compromise neural network integrity by injecting errors into DNN parameters (i.e., weights), causing sizable accuracy drop. Early attacks on DNNs with floating-point weights tend to flip the most significant bits (MSBs) of the exponent to cause a huge deviation in the weight values [11]–[13]. Recently, a new type of Bit-Flip Attack (BFA) was shown to be able to significantly degrade the accuracy of a quantized DNN models with a tiny set of bit-flips [14]. A following work developed a stealthy 1-to-1 Targeted Bit-Flip Attack (T-BFA) wherein only the inputs from one source class are hijacked to one target class [16].

The main differences between BFA and T-BFA lie in their objectives. Within minimum number of bit-flips, BFA aims at maximizing the inference loss w.r.t true labels so as to crash the DNN, while T-BFA aims to classify inputs from a specific source class p ($\mathbb{X}_p$) as a target class q without affecting the classification of the remaining inputs. The objectives of BFA and TBFA are respectively formulated in Eqns (1) and (2):

$$\max_{\{\hat{B}\}} \mathbb{E}_x\{\mathcal{L}\left(f(x,\{\hat{B}\})\right),t\} \tag{1}$$

$$\begin{aligned}\min_{\{\hat{B}\}} \mathbb{E}_x\{&\mathcal{L}\left(f(x,\{\hat{B}\}),t_q\right)|x\in\mathbb{X}_p\\ &+\mathcal{L}\left(f(x,\{\hat{B}\}),t\right)|x\in\mathbb{X}_j\}\end{aligned} \tag{2}$$

where $\{\hat{B}\}$ is the quantized representation of the DNN's weights in 2's complements form after fault injection, x and t are respectively the input data and corresponding ground-truth label, $f(x,\{\hat{B}\})$ computes the DNN output for input x, $\mathcal{L}\left(.,.\right)$ denotes the cross-entropy loss between the DNN output and the ground-truth label, j is a class other than the source class p.

Despite their different objectives, both BFA and T-BFA follow the same iterative gradient-based Progressive Bit Search (PBS) process with subtle distinctions. In each iteration, PBS randomly selects a small set of training images and uses them to identify one bit to flip in two steps: *intra-layer search* wherein one candidate weight bit is identified for each layer of the target DNN, and *inter-layer selection* which compares candidate bits across all layers and flips the best one. In the kth iteration, the intra-layer search ranks a bit b_i by the absolute value of its gradient w.r.t. loss (i.e., $|\partial\mathcal{L}/\partial b_i|$) and selects the top n bits with highest gradients. It then flips these n bits one at a time and records the loss increment. To meet the objectives of BFA/T-BFA, bits are flipped in the opposite/same direction of gradient to respectively maximize/minimize the loss. Subsequently, the inter-layer search of BFA/T-BFA identifies, across all L layers, the bit with max/min loss from the loss set $\{\mathcal{L}_1^k,\mathcal{L}_2^k,...,\mathcal{L}_L^k\}$ and flips it. This iterative search process terminates when the attack goal is achieved.

BFA in real platforms: The work in [15] demonstrates the feasibility of launching bit-flip attacks in Ivy Bridge-based Intel i7-3770 CPU with two memory channels via performing precise rowhammer attacks named *Deep-Hammer*. Unlike standard rowhammer attacks that flip

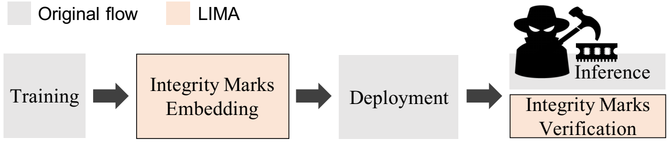


Fig. 1. LIMA overview

bits probabilistically, DeepHammer needs to flip the chain of target bits identified by BFA *precisely* without altering the remaining bits. This is achieved via a precise double-sided hammering using targeted column-page-stripe. The preciseness is guaranteed by allowing only one bit-flip per memory page [15].

Countermeasures against BFAs: As BFA is a new type of fault injection attacks, very few countermeasures have been proposed. One idea is to design a binarization-aware DNN and its relaxation (called *piece-wise clustering*) of the DNN weight parameters [29], which retrains the DNN to reduce the number of close-to-zero weights (which are most vulnerable to BFA). Another retraining-based work [30] reconstructs DNN weights in a way that the amount of weight perturbation caused by BFA is minimized or diffused to neighboring weights. While these two techniques provide full coverage to all the layers of a target DNN, they require expensive retraining. They also cause noticeable change to the weight distribution which not only make the defense easily detectable by the attacker, but also induce non-trivial accuracy drop and thus are only effective for small networks and datasets. As for large and complicated networks and datasets (e.g. ImageNet), it is difficult to balance the improvement in resilience and the accuracy drop caused by these countermeasures [31].

Recently, a BFA detection approach is presented in [31] whose fundamental goal is to differentiate BFAs from random bit errors. The work observes that most of the bit-flips selected by BFA are clustered within a few layers, and therefore proposes to embed, in those layers, certain watermarks sensitive to BFAs but not to random errors. Unfortunately, this work makes an unrealistic assumption of random bit errors only occurring in the lower-order bits of weights. Furthermore, an attacker with knowledge of this countermeasure can bypass it by simply injecting faults into those unprotected layers.

III. Threat Model

We follow the same threat model used by the previous BFA-related work [15], [16], [31] for compromising the integrity of a quantized DNN model. BFA is a white-box attack, i.e., the attacker is assumed to have complete knowledge of the target DNN structure, weights, and gradients, and partial knowledge of the training data set. Such knowledge of the model can be obtained via side channel attacks, as have been demonstrated in [32], [33]. The attacker program and the victim DNN model are assumed to co-locate on a machine equipped with DRAM memory, a typical setting in machine-learning-as-a-service (MLaaS) platforms. The victim DNN model is quantized into 2's complement form and all of its weights, bias, and batch normalization parameters are stored in the DRAM memory of the target platform where it runs. The attacker pre-selects target bit-flip locations by performing gradient-based BFA technique [14], and performs DeepHammer attack (i.e., precise double-sided hamming) to flip the chain of bits selected by BFA [15]. The defender is assumed to have access to the target DNN structure and weights (which are trained and quantized already) but not to any part of the training dataset.

IV. LIMA Framework

A. Framework Overview

Figure 1 illustrates an overview of LIMA's two-step BFA mitigation process: (1) embedding integrity marks into DNN weights before its deployment, and (2) verifying them at the inference stage to capture BFAs at a testing period chosen by the user.

LIMA's integrity marks are embedded into every single DNN layers by establishing certain property between the layer's weights via slightly modifying DNN weight values (± 1 difference), without causing any noticeable accuracy loss. Verifying these integrity marks in the computed intermediate feature maps makes the faults injection attacks observable in the output of all DNN layers, including hidden layers which are not supported by testing-based schemes. This is done without the need for saving any intermediate feature maps, a significant benefit as the size of feature maps could be much larger than that of model parameters [34]. Meanwhile, LIMA achieves full controllability of the hidden layers with a special *Read-Check-Write (RCW)* process (Fig. 2), added in between two consecutive layers l_i and l_{i+1} to read the content of the intermediate data (h_i), check the integrity property to capture BFA, and write the test inputs for checking the integrity of the next layer. RCW is implemented in software, and is only activated in the BFA detection mode, thus imposing zero overhead to the original inference process. Last but not the least, the integrity marks allow for not only the *detection* but also the *location* of BFAs within a group of weights and a

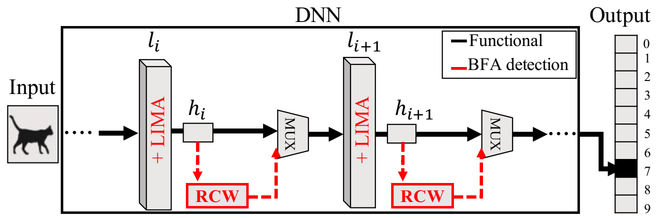


Fig. 2. LIMA's defense paradigm. Light-weight integrity properties are embedded into every DNN layer (l_i), a Read-Check-Write(RCW) process is added in between layers to read the intermediate data (h_i), check the integrity property, and prepare test inputs for the next layer.

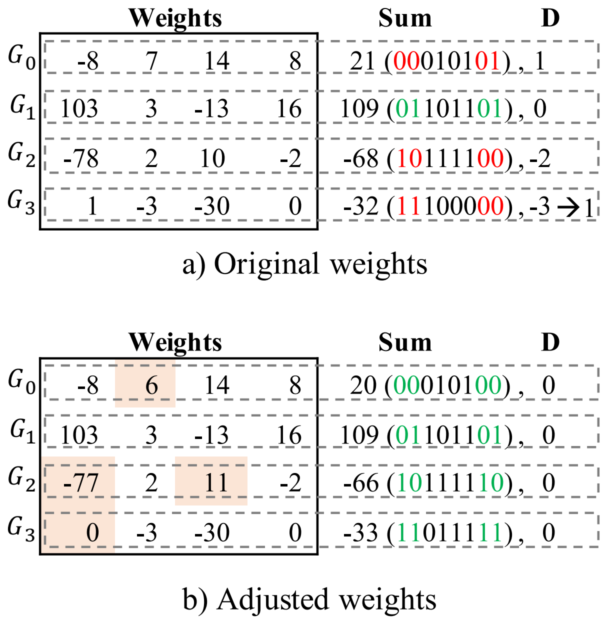
subsequent cost-effective run-time *recovery* from BFA, as will be shown later.

B. Integrity Marks Embedding

A high quality integrity mark should be easy to establish and verify, and sensitive enough to capture BFAs while inducing no inference accuracy drop. Instead of using traditional fault detection properties (such as ECC, parity bits, and checksums) that uniformly cover all bit positions and hence impose non-trivial overhead, our fundamental idea is to *protect each weight, but only the bits that are most vulnerable to BFAs*. Our extensive studies on BFA show that most of the time the attack flips the Most Significant Bits (MSBs) of weight values while Least Significant Bits (LSBs) are almost never selected[1]. Accordingly, we propose to *establish a property between the LSBs and MSBs*. As LSBs are much less critical, slightly modifying them will not cause unacceptable accuracy drop. To further minimize the potential accuracy drop, we propose to establish the property in *the sum of a group of DNN weights* in each layer. This also means that without the exact group information, an attacker cannot verify the property or flip two bits in a group to bypass the verification.

Definition IV.1 (LIMA's integrity property)**.** Each DNN layer is partitioned into groups of size n, and the K-bit MSBs and LSBs of the sum of the n weights in each group should be identical.

If the sum of a group of weights does not satisfy this condition, the property can be established by slightly modifying the weight's LSBs. Fig. 3 illustrates an example of embedding a K=2-bit integrity marks in a 4×4 weight matrix where every four weights in a row are grouped together. Fig. 3(a) shows the row-wise sum of the original weight matrix. Only G_1 has identical 2-bit MSBs and LSBs of its sum value, while the sum of G_0, G_2, and G_3 should be changed by 1, -2, and 1, respectively, to form this property. In Fig. 3(b), after incrementing/decrementing the marked weight values, the property holds for all four groups.



	Weights				Sum	D
G_0	-8	7	14	8	21 (00010101)	1
G_1	103	3	-13	16	109 (01101101)	0
G_2	-78	2	10	-2	-68 (10111100)	-2
G_3	1	-3	-30	0	-32 (11100000)	-3 → 1

a) Original weights

	Weights				Sum	D
G_0	-8	6	14	8	20 (00010100)	0
G_1	103	3	-13	16	109 (01101101)	0
G_2	-77	2	11	-2	-66 (10111110)	0
G_3	0	-3	-30	0	-33 (11011111)	0

b) Adjusted weights

Fig. 3. Embed 2-bit integrity property in four groups of weights.

Algorithm 1 shows how the K-bit integrity property is embedded into a DNN layer with weight matrix W under a specific grouping and candidate selection policy. The process starts by finding all the groups G inside the layer, calculating the sum S of all the weights in G, and finding the difference D between K-bit MSBs and LSBs of S (lines 1–5). The remaining steps of the algorithm reduce this difference to zero via inserting a minimal deviation of ± 1 into a set of $|D|$ weights that are chosen according to the selection policy (lines 6– 13). To further reduce the number of weights to be modified, our algorithm reverses the direction of weight adjustment if the difference between LSBs and MSBs is bigger than 2^{K-1} (lines 8–10). This is shown in the case of G_3 in Fig. 3(b), where the value of D is set to $-3+2^2=1$, the direction of modification is changed from add to sub, and one (instead of three) weight is modified. Note that this modification also generates a carry of -1, which changes the four middle bits of sum of G_3 from 1000 to 0111. If this carry propagates to the MSBs and causes MSBs $\neq$ LSBs, an extra ± 1 is made to one more weight in the group (lines 12 – 13). In sum, the maximum number of bits modified to establish a K-bit property is limited to 2^{K-1}.

[1] This observation is in line with findings in [29], [31], and is also expected since faults in MSBs cause the largest deviations in weight values and hence degrade DNN accuracy most severely.

Algorithm 1 Integrity property establishment

```
Require:
    weight-matrix $W$, property-width $K$,
    grouping-policy $GP$, selection-policy $SP$
$groups \leftarrow GP(W)$
for all $G$ in $groups$ do
    $S \leftarrow sum(G)$
    $M, L \leftarrow K$ MSBs and LSBs of $S$
    $D \leftarrow L - M$
    $reversed \leftarrow False$
    if $D \neq 0$ then
        if $|D| > 2^{(K-1)}$ then
            $D \leftarrow \begin{cases} D + 2^K, & \text{if } D < 0 \\ D - 2^K, & \text{if } D > 0 \end{cases}$
            $reversed \leftarrow True$
        Subtract $D/|D|$ from $SP(G, |D|)$
        if $reversed$ and property does not hold then
            Subtract $D/|D|$ from $SP(G, 1)$
```

C. Integrity Marks Design Space

Design space of the proposed integrity property involves configuring three input parameters of Algorithm 1, namely, the property width K, the candidate weight selection strategy, and the grouping policy.

If the property width is minimum ($K = 1$), only the MSB (i.e., the sign bit) is protected. Flipping the sign bit of an integer does not necessarily cause the largest gradient w.r.t. loss and hence is not always selected by BFA. Enlarging the property width, on the other hand, protects more bits but induces higher accuracy drop potentially. To select the best value of K, we ran BFA on three 8-bit DNN models: Googlenet and Resnet-18 on Imagenet dataset, and VGG-16 on CIFAR-10 dataset, and collected the positions of bit-flips. Our statistics show that more than 95% of bit-flips are in the two MSBs (b_7 and b_6). Hence, $K = 2$ seems to be a viable option for most DNNs to balance the level of security enhancement and potential accuracy drop.

There are multiple possible ways to select the candidate weights for modification, such as always selecting weights with the largest (or smallest) absolute values, or selecting them randomly. Our study shows that their impact on the DNN accuracy is negligible, which stems from the fact that the weight modification for embedding the proposed integrity property is already minimal. As a result, LIMA adopts a *random* selection strategy as it reduces the complexity of the weight selection process while at the same time adding an extra level of protection since the locations of the modified weights are unpredictable and hence unknown to the attacker.

In terms of the grouping policy, LIMA offers the flexibility of grouping weights in different granularities and also across various dimensions. The group granularity is expected to affect inference accuracy: the smaller the group size is, the more the ratio of weights that need to be modified to establish the integrity property, and hence a more noticeable accuracy drop is expected. Enlarging the group size, on the other hand, reduces the impact on DNN accuracy but may increase the chance of false negatives, which may occur if the attacker flips two bits of different weights in one group.

Under the same granularity, various grouping methods across multiple dimensions can be adopted. These different grouping methods are not expected to have a noticeable impact on the DNN accuracy or the ability to detect BFAs, but rather add another level of security protection. Specifically, if a skilled attacker wants to bypass LIMA defense by injecting pair-wise bit-flips within a group, the attacker needs to find out the exact grouping method. Assuming that G and W respectively denote the group granularity and the total number of weights in a DNN layer, LIMA's groups can be formed with weights $[i, i+S, \ldots, (i+mS)\%W]$ where S is the grouping stride, and there are W-1 possible strides and G possible starting points for a group with G elements. To find the exact grouping method, the attacker has to brute force a search space of $G * (W - 1)$ for every possible G values that divides W.

D. Integrity Marks Verification

One advantage of the proposed integrity property is that it can be verified for online BFA detection with standard convolution operations, requiring no extra hardware support. Specifically, the sum of the group of weights can be calculated by *setting a region of inputs that will be convolved with the target group of weights to 1*. As an example, Fig. 4 illustrates how the inputs of a convolution layer is modified to compute the sum of weights in filter A. The inputs are in three channels: red, green, and blue, which are convolved with a 3-D 3×3×3 kernel. By setting 9 inputs in the red channel to 1 and all inputs in the green and blue channels to 0, the sum of all weights in filter A can be calculated. This process is marked with ① in Fig. 4. As the location of the target sum in the output feature map is known beforehand ($\sum A$ in Fig. 4), the RCW procedure shown in Fig. 2 can read it and verify if the integrity property holds or not. If a large group size is adopted such that one group contains multiple filters (e.g., A and B), the sum can be computed by setting the inputs of multiple channels (e.g., red and green) to 1. On the other hand, if the group size

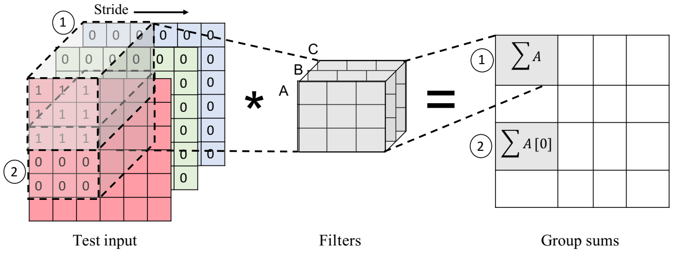


Fig. 4. Verifying integrity property with convolution operations

TABLE I
DATASETS AND DNN MODELS' SPECIFICATIONS

Dataset	DNN	Layer #	Weight #	Accuracy
Imagenet	Googlenet	22	$6.6M$	69.30%
Imagenet	Resnet18	18	$11M$	69.22%
CIFAR-10	VGG-16	16	$15M$	90.88%

TABLE II
SAMPLE WEIGHT GROUPING POLICIES EVALUATED

Kernel size	Group size (# weights)		
	Small	Medium	Large
7×7	7	49	147
3×3	9	36	144
1×1 (FC)	8	32	128

is smaller than the filter size, the convolution stride can be leveraged to compute a partial sum of weights. For example, shifting filter A two steps to the right computes the sum of the leftmost column of A, and shifting A two steps down computes the sum of its top row which is marked with ② in Fig. 4.

For each convolutional layer, RCM writes test inputs in a way that the output feature map contains different partial sum values, while the RCW procedure at the layer's output can selectively verify only the target integrity property (e.g., $\sum A$ or $\sum A[0]$ in Fig. 4). For fully connected layers, since the computation of output neuron values (a special type of output feature maps) can be treated as 1×1 convolution operations, by directly leveraging MAC operations, one can follow the same approach for property verification.

With the verification process outlined above, LIMA can quickly detect and locate BFAs in each layer within a group of weights. Unlike inference, there is no dependency between the current layer's output and the next layer's input when verifying the integrity marks, implying that checking of all DNN layers can be done in parallel. Overall, a BFA detection procedure is involved in between two inference runs. Faults injected into the to-be-checked groups will be detected immediately, while faults injected into the already-checked groups will be captured in the next round of detection.

E. BFA Detection Accuracy Analysis

The proposed integrity property always holds if there is no bit-flip in the weight values, and thus, it does not have any *false positives* (unless a non-malicious random fault has occurred in the LSBs or MSBs, which is outside of the scope of this work). However, there are a few possible cases that may lead to *false negatives*. First, if the attacker chooses to flip an unprotected bit not located in the K-bit MSBs, the bit-flip will be detected only if it generates a carry and propagates to the K-bit MSBs in the group sum. Second, if the attacker flips bit b_i of two weights in the same group but in opposite directions (one 1→0 and the other 0→1), the sum will not change, and the bit-flips will not be detected. Likewise, if the attacker flips bit b_i of 2^n weights in the same group in the same direction, it is equivalent to flipping bit b_{i+n} in one weight. If $i+n \geq q$ (q is the quantization level), the bit-flips will not alter the q-bit sum and hence will not be detected. Finally, if the attacker flips an MSB b_i and a corresponding LSB ($b_i + K - Q$) in the same group in the same direction (e.g., bits b_7 and b_1), the sum value changes but the integrity property still holds and thus, the attack will not be detected. Although theoretically possible, this case practically has never happened as the PBS process of BFA never selects LSBs to flip.

The aforementioned false negative cases are rare unless an attacker modifies the BFA procedure to intentionally bypass LIMA. Furthermore, although it is possible for a few bit flips to escape detection, as BFA needs to inject a chain of bit-flips to crash a network, it is extremely unlikely to have all the bit-flips in the chain be false negatives and the BFA undetected.

V. EXPERIMENTAL EVALUATION

A. Experimental Setup

LIMA is evaluated on two well-known image recognition datasets: CIFAR-10 [35] containing 60K color images in 10 classes, and Imagenet [36] containing 1.35M color images in 1000 classes. These datasets are trained with three popular DNN models: Googlenet [37], Resnet-18 [38], and VGG-16 [39]. Table I shows the number of layers, weight count, and the original accuracy of the 8-bit quantized implementations of these models.

BFA was performed on the selected DNNs with attack sample size of 128 images, and terminated when DNN's accuracy drops to the random guess level, i.e., 10% for CIFAR-10 and 0.1% for Imagenet. For each set of experiments, 20 instances of BFAs (identified with 20

TABLE III
IMPACT ON DNN INFERENCE ACCURACY

DNN	w/o LIMA	w/ LIMA		
		Small	Medium	Large
Googlenet	69.30%	68.88%	69.15%	69.19%
Resnet-18	69.22%	69.12%	69.13%	69.16%
VGG-16	90.88%	90.91%	90.85%	90.90%

TABLE IV
AVERAGE NUMBER OF BIT-FLIPS SELECTED BY BFA, BEFORE AND AFTER EMBEDDING INTEGRITY MARKS

DNN	w/o LIMA	w/ LIMA		
		Small	Medium	Large
Googlenet	6.0	6.6	6.7	6.9
Resnet-18	12.0	10.9	11.1	12.9
VGG-16	40.0	54.8	47.6	38.0

TABLE V
LIMA'S DETECTION CAPABILITY FOR BFA CHAINS AND INDIVIDUAL BIT-FLIPS

DNN	Small group		Medium group		Large group	
	Chain	Single	Chain	Single	Chain	Single
Googlenet	100%	99.5%	100%	87.7%	100%	86.8%
Resnet-18	100%	90.2%	100%	77.5%	100%	70.5%
VGG-16	100%	78.3%	100%	78.0%	100%	73.2%
Average	100%	89.3%	100%	81.1%	100%	76.8%

different random seeds) were executed. The same seeds were used across all the experiments conducted under different configurations to ensure fairness in comparison.

LIMA was implemented as a reconfigurable extension to the BFA tool [14] in PyTorch. For all the experiments, the property width was set to K=2 and candidate weights for modification are selected randomly. A row-wise grouping strategy was adopted, and three sample grouping granularities (small, medium, and large) were evaluated as test cases. Table II shows the group sizes for different kernel shapes used by all three DNN models. Note that policy in the last row is applied on both kernel size 1×1 and fully connected (FC) layers.

LIMA was run on a Google Colab Pro server with 25GB memory and T4/P100 GPUs. Embedding the proposed integrity marks in the entire network only causes a one-time overhead of 15–150 seconds, confirming that it can be easily added to a trained and quantized DNN.

B. Impact on DNN Accuracy

Tables III reports the impact of embedding the proposed integrity property on DNNs' inference accuracy, which is close-to-zero across all cases. The maximum accuracy drop for Googlenet, Resnet-18, and VGG-16 are 0.42%, 0.1%, and 0.03%, respectively. For Googlenet and Resnet-18, Large group size causes the least accuracy drop and small group size causes the most, which is consistent with the theoretical discussion in Section IV-C. However, this trend does not hold for VGG-16 which, trained on small-scale CIFAR-10 dataset containing only 10 classes, has very high original accuracy. As a result, the accuracy change caused by the integrity property is so small that the impact of randomness in seed selection becomes dominant. Overall, the results demonstrate that the grouping policy does not have a remarkable impact on inference accuracy.

Table IV reports the number of bit-flips required by BFA to crash the selected DNN models. For Googlenet and Resnet-18 which are trained on Imagenet, BFA can crash the original model with only 6 and 12 bit-flips, respectively. In comparison, VGG-16, as it is the largest network and is used to classify small-scale CIFAR-10 images, carries much more redundancy hence is more resilient to BFA. The same experiments are repeated on models embedded with LIMA (since it slightly adjusts weight values). As shown, increasing the group size makes the models trained on Imagenet slightly less susceptible to BFA but for CIFAR-10 the trend is opposite. Overall, the results confirm that the impact of LIMA on a model's behavior against BFA is insignificant.

C. BFA Detection Capability

Since BFA requires flipping a chain of pre-selected bit-flips, it can be detected if any of the flipped bits are captured. Based on this observation, Table V reports the detection rates of both BFA chains and individual bit-flips. The results show that LIMA can detect, with 100% accuracy, all the bit-flip chains inserted in DNNs under all the grouping configurations. In terms of individual bit-flips, it is able to detect 70.5%–99.5% of them. As expected, the smaller the group size, the lower the false negative rate of detecting individual bit-flips. On average, 10.7%, 18.9%, and 23.2% of bit-flips in Small, Medium, and Large grouping policies end up becoming false negatives. The three possible false negative cases are discussed in Section IV-E.

Overall, our experiments confirm that LIMA can 100% detect bit-flip attacks. If the defender aims to not only detect the attack but also precisely locate individual bit-flips, a smaller group size is preferred.

VI. DISCUSSION

To evaluate whether LIMA qualifies to be a high-quality defense scheme, this section presents an in-depth study on LIMA's capability to detect more stealthy attacks as well as its stealthiness, robustness against

TABLE VI
AVERAGE NUMBER OF BIT-FLIPS SELECTED BY T-BFA AND LIMA'S DETECTION CAPABILITY

Scheme	Number of bit flips	Detected	
		Chain	Single
w/o LIMA	11.0	–	–
w/ LIMA-Small	11.7	100%	98.3%
w/ LIMA-Medium	11.7	100%	94.4%
w/ LIMA-Large	11.9	100%	81.9%

bypassing attempts, and potential for guiding recovery process. The experiments are performed on Resnet-18 – the largest DNN model evaluated on the larger Imagenet dataset.

A. Robustness against Stealthy BFA Attacks

While LIMA's detection capability against BFA has been demonstrated, one question remaining open is whether it can detect stealthy variations of BFA attacks that can escape input-based testing [18], [20], including both T-BFA and early termination of BFA (before the DNN accuracy drops to random guess), which disrupts the DNN's functionality with fewer bit-flips and potentially has a higher chance of escape detection.

First, we implemented the T-BFA attack [16] which alters the output of a source class p to a target class q while retaining the classification of other classes as much as possible. Using the same seeds as the ones used for BFA, 20 instances of T-BFAs were executed for each of the three group sizes. Then the number of bit flips was collected for both the original (unprotected) network and the ones with LIMA embedded. Classes p and q were randomly selected in each run. Each run was terminated when at least 98% of images in class p's were changed to class q. At that time, the accuracy of the network slightly dropped (minimum of 50%).

Table VI shows the impact of LIMA on the number of bit-flips selected by T-BFA, as well as its detection capability. Same as what was observed for BFAs, embedding LIMA does not cause any remarkable impact on the total number of bit-flips required to conduct the attack. Meanwhile, LIMA can successfully detect 100% of T-BFA attack chains and at least 81.9% of single bit-flips. This strong detection capability stems from the fact that despite having different loss functions and goals, both BFA and T-BFA follow a grandient-based approach to select vulnerable bits. MSBs which typically show higher gradient values are most likely to be selected in both attacks. A closer look into the positions of the injected bit-flips confirms this observation: almost all faults in T-BFAs are injected into the two highest-order bits.

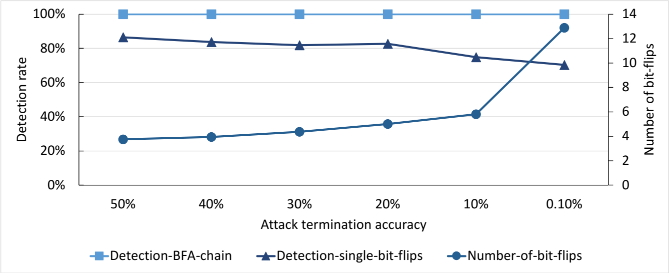


Fig. 5. Impact of attack termination accuracy on the number of bit-flips needed by BFA and LIMA's chain and single bit-flip detection rates under Large grouping policy.

Second, to study LIMA's robustness against early terminated BFAs, we evaluated the impact of various attack termination criteria on the number of BFA bit-flips as well as LIMA's detection rate. Figure 5 shows the results under the Large group size which offers the lowest detection capability, which confirm the excellent detection capability of LIMA. First, it is able to 100% detect BFA chains even under a very early termination accuracy of 50%. Furthermore, the higher the attack termination accuracy, the shorter the length of BFA chain, the higher the detection rate of individual bit-flips. The majority of false negative cases occur when the termination accuracy is dropped below 20%. This is because BFA follows a greedy algorithm to find the most vulnerable bits in DNNs. Hence, the first few bit-flips are usually injected into weights scattered across different groups, making them 100% detectable by LIMA.

Overall, the results confirm LIMA as an effective and extensive defense mechanism against the stealthy variations of BFA attacks.

B. Impact on DNN Weight Distribution

A high-quality defense scheme should not leave any observable footprint on the DNN model. As LIMA requires slight modifications to the weight values, we evaluated whether LIMA causes a noticeable change to the weight distribution that an attacker can tell. Fig. 6 compares the weight distribution of Resnet-18 before and after embedding LIMA. The comparison is between the original model and the LIMA version with Small grouping policy as it modifies the largest ratio of weights (our statistics show that the Small, Medium, and Large group sizes modify 11%, 2.8%, and 0.7% of weights, respectively). As shown, because the modifications to weight values are minimum (±1), there is no noticeable change in the DNN weight distribution even when 11% of weights are adjusted. This also ensures that LIMA incurs very marginal accuracy change to the original

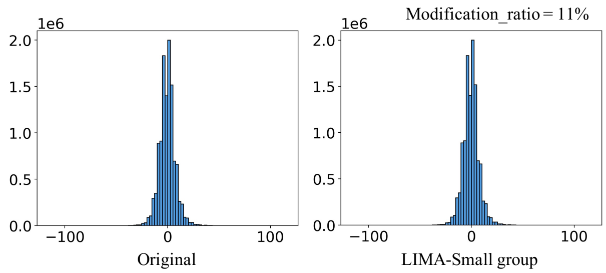


Fig. 6. Negligible impact of LIMA on DNN weight distribution of Resnet-18

model as shown in Tables III. Therefore, attackers cannot easily tell whether LIMA is embedded in a DNN model.

C. Robustness against Bypassing

Regardless of its level of stealthiness, there is always a chance for the attacker to know the countermeasure precisely. For this reason, we evaluated LIMA's robustness against bypassing attempts under a possible (but highly unlikely) scenario where the attacker has prior knowledge of LIMA and its major design features including the property width K, the group size G, and even the exact group partition. Under this assumption, the attacker may take three routes to bypass LIMA, denoted as *B1*, *B2*, and *B3*. *B1* avoids injecting bit-flips in the protected K-bit MSBs and targets the unprotected lower-order bit positions. As flipping the lower-order bits causes smaller value deviations, more bit-flips are required to crash the DNN. In comparison, *B2* and *B3* inject bit-flips in pair so as to maintain the integrity property. Specifically, for a most vulnerable bit flipped in position b_i, *B2* flips an extra bit in the same MSB position of another weight in the same group but in the opposite direction, while *B3* flips the corresponding LSB (position $b_i + K - Q$) of a weight in the same group in the same direction. These approaches double the complexity of BFA and require knowledge of the exact grouping strategy, which is difficult to obtain given the high number of grouping possibilities described in Section IV-C.

The aforementioned bypassing methods were implemented and evaluated under two extreme grouping policies: Small and Large. Meanwhile, to compare with previous work, we also implemented the watermark-based BFA detection [31] and a straightforward bypassing technique that injects bit-flips only in layers not protected with watermarks. All schemes were evaluated under the constraint imposed by DeepHammer [15], the work that demonstrated feasible BFAs in real platforms by restricting only one precise bit-flip per page (without altering other bits). As this strict constraint sometimes prevents us finding a valid attack, we relaxed it by allowing up to two precise bit-flips per page in experiments. Furthermore, to prevent the BFA algorithm from being stuck in an endless loop, we added two extra termination conditions: (1) if the total number of bit-flips injected is $100\times$ what was required for breaking the original model (i.e., more than $12 \times 100 = 1200$ bit-flips for Resnet-18); (2) if, for five consecutive iterations with different sets of image samples, the attacker was not able to find a bit to flip. Attacks terminating in these two conditions are marked as *unsuccessful*.

Table VII presents the ratio of successful attacks, their number of bit-flips required to break Resnet-18, and their ability to bypass the underlying countermeasure. *Baseline* denotes the Rowhammer-based BFA attack on an unprotected DNN, while *1-layer* and *2-layers* are two versions of the watermark scheme [31] which protects only one and two DNN layers.

The results reveal several interesting points. First, the number of bit-flips in the baseline confirms that the strict DeepHammer constraint makes it more difficult to launch a BFA. With 1 and 2 bit-flips permitted per page, BFA requires an average of 22.6 and 14.2 bit-flips, both of which are higher than the 12 bit-flips needed under no constraint (reported in Table IV). These numbers are used as the reference for computing the complexity of bypassing techniques, e.g., an unsuccessful attack that requires more than 1200 bit-flips has a complexity of $1200/22.6$= 53.10× and $1200/14.2$=84.51× when respectively 1 and 2 bit-flips per page are allowed. Second, the watermarking scheme cannot effectively prevent bypassing attempts or increase the complexity of BFA; by injecting faults in the unprotected layers, the model crashes almost within the same number of bit-flips. Protecting two layers only slightly increases the attack complexity by 1.02–1.03×. In comparison, LIMA can completely counteract all the *B1*, *B2*, and *B3* attempts (0% success rate) when only one bit-flip per page is allowed. When the constraint is relaxed to two bit-flips per page, LIMA still counteracts *B2* and *B3* attempts for Small group size. The fundamental reason for these unsuccessful attempts is that they require more bit-flips to bring the DNN accuracy down to random guess, while the DeepHammer constraint only allows very limited number of bit-flips per page. A further examination shows that at the time of termination, the unsuccessful *B1*, *B2*, and *B3* attempts were able to reduce the DNN accuracy to around 0.39%, 22.69%, and 0.5% respectively, indicating that it is relatively easier for *B1* and *B3* to find a qualified bit to flip. In case of

TABLE VII
Robustness of LIMA and watermarking [31] against bypassing techniques

Scheme	1 bit-flip per page				2 bit-flip per page			
	Success rate	bit-flips	Complexity	Bypassed?	Success rate	bit-flips	Complexity	Bypassed?
Baseline	100%	22.6	1.00×	–	100%	14.2	1.00×	–
Watermark 1-layer	100%	22.8	1.01×	100%	100%	14.5	1.02×	100%
Watermark 2-layers	100%	23.0	1.02×	100%	100%	14.6	1.03×	100%
LIMA-B1-Small	0%	> 1200	> 53.10×	0%	100%	250.2	17.62×	0%
LIMA-B1-Large	0%	> 1200	> 53.10×	0%	100%	215.2	15.16×	0%
LIMA-B2-Small	0%	> 1200	> 53.10×	0%	0%	> 1200	> 84.51×	0%
LIMA-B2-Large	0%	> 1200	> 53.10×	0%	100%	251.5	17.71×	100%
LIMA-B3-Small	0%	> 1200	> 53.10×	0%	0%	> 1200	> 84.51×	0%
LIMA-B3-Large	0%	> 1200	> 53.10×	0%	100%	104.5	7.36×	100%

the successful *B1* attempts, LIMA is still capable of detecting them. This is because faults injected into the lower-order unprotected bits may cause the generation of a carry in the sum value that propagates to the protected MSBs and is caught by LIMA. Overall, *B2* and *B3* are the only effective approach for successfully launching BFA and bypassing LIMA, which is only possible under relaxed constraints, i.e., the Large group partitions are used and two bit-flips per page are allowed. Even in that case, the attacker's effort increases at least by 17.71× and 7.36× for *B2* and *B3*, respectively. If the defender aims to prevent *B2* attempts, using a smaller group size is a viable solution.

Compared with related work, LIMA's ability in increasing BFA attack complexity is significantly higher than the previous weight-clustering approach [29], which increases the number of bit-flips only by 2×, at the cost of performing expensive retraining and imposing more than 2% accuracy drop.

D. Recovery from BFA

Once LIMA captures BFAs, a recovery process can be initiated to rescue DNN accuracy. The ultimate recovery solution is to reload all DNN weights. As LIMA provides precise faulty group information, it significantly reduces recovery overhead since only the groups of faulty weights (instead of the entire model) need to be reloaded.

For real-time systems such as autonomous vehicles where an important decision must be made in time, costly model transmitting and reloading may cause the system to miss the deadline. In this case, LIMA's precise faulty group information enables rapid on-line recovery as a temporary damage control mechanism. The ideal on-line recovery is to cancel out the effect of the injected faults by finding the exact locations of bit-flips and flip them back. However, as LIMA detects faults in the granularity of a group, neither the faulty weight nor the faulty bit position is known. While it is possible to develop heuristics to *guess* such information based on the differences between MSBs and LSBs of the faulty group's sum value, such a guess could lead to more adverse impact if it is incorrect. As an alternative, a conservative recovery strategy is to set all weights within the faulty group to zero to eliminate the impact of faulty weights. This approach is motivated by the observation that most DNN weight values are close to zero, and is in-line with the approach used in [40] which detects faults at the granularity of neurons and corrects them by setting the neuron output to zero. Clearly, this recovery scheme is expected to be more effective for smaller group sizes. More precisely, our experiments show that for Resnet-18, this approach can successfully rescue the accuracy up to 68.96% when Small group size is used. Hence, if the accuracy of on-line recovery process is important to the defender, small group sizes are more favorable.

VII. Conclusions

This paper presented a LIMA, a light-weight countermeasure against Bit-Flip Attacks (BFAs), which protects DNN integrity by embedding integrity properties into the sum of a group of weights. The countermeasure can be applied to a trained and quantized DNN model without demanding any extra hardware or storage support, and can be easily verified during the inference stage to capture potential BFAs. Our comprehensive experimental study shows that LIMA is capable of detecting 100% of BFA and T-BFA bit-flip chains while causing near-zero accuracy loss. It is undetectable by attackers and is extremely difficult to bypass. LIMA also offers a rich design space for the defender. Without affecting its detection capability, the grouping configuration of LIMA can be tuned to lower the implementation overhead or increase bypassing difficulty and recovery accuracy.

References

[1] Y. Liu, A. Jain, C. Eng, D. H. Way, K. Lee, P. Bui, K. Kanada, G. de Oliveira Marinho, J. Gallegos, S. Gabriele *et al.*, “A deep learning system for differential diagnosis of skin diseases,” *Nature Medicine*, pp. 1–9, 2020.

[2] T. Schaffter, D. S. Buist, C. I. Lee, Y. Nikulin, D. Ribli, Y. Guan, W. Lotter, Z. Jie, H. Du, S. Wang *et al.*, “Evaluation of combined artificial intelligence and radiologist assessment to interpret screening mammograms,” *JAMA network open*, vol. 3, no. 3, pp. e200 265–e200 265, 2020.

[3] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, “Multinet: Real-time joint semantic reasoning for autonomous driving,” in *IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1013–1020.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[6] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2574–2582.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[8] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *arXiv preprint arXiv:1611.01236*, 2016.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *IEEE European symposium on security and privacy (EuroS&P)*, 2016, pp. 372–387.

[10] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, “Ead: elastic-net attacks to deep neural networks via adversarial examples,” *arXiv preprint arXiv:1709.04114*, 2017.

[11] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault injection attack on deep neural network. in 2017 ieee,” in *ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138.

[12] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, “Practical fault attack on deep neural networks,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 2204–2206.

[13] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, “Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 497–514.

[14] A. S. Rakin, Z. He, and D. Fan, “Bit-flip attack: Crushing neural network with progressive bit search,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[15] F. Yao, A. S. Rakin, and D. Fan, “Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1463–1480.

[16] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, “T-bfa: Targeted bit-flip adversarial weight attack,” *arXiv preprint arXiv:2007.12336*, 2020.

[17] M. Ribeiro, K. Grolinger, and M. A. Capretz, “Mlaas: Machine learning as a service,” in *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, pp. 896–902.

[18] Q. Liu, T. Liu, Z. Liu, W. Wen, and C. Yang, “Monitoring the health of emerging neural network accelerators with cost-effective concurrent test,” in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[19] Z. He, T. Zhang, and R. Lee, “Sensitive-sample fingerprinting of deep neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4729–4737.

[20] F. Meng, F. S. Hosseini, and C. Yang, “A self-test framework for detecting fault-induced accuracy drop in neural network accelerators,” in *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 722–727.

[21] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O’Connell, W. Schoechl, and Y. Yarom, “Another flip in the wall of rowhammer defenses,” in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 245–261.

[22] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, “Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks,” in *IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 55–71.

[23] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,” *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[24] M. Son, H. Park, J. Ahn, and S. Yoo, “Making dram stronger against row hammering,” in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[25] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, “Mitigating wordline crosstalk using adaptive trees of counters,” in *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 612–623.

[26] J. M. You and J.-S. Yang, “Mrloc: Mitigating row-hammering based on memory locality,” in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[27] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, “Twice: preventing row-hammering by exploiting time window counters,” in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019, pp. 385–396.

[28] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, “Graphene: Strong yet lightweight row hammer protection,” in *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1–13.

[29] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, “Defending and harnessing the bit-flip based adversarial weight attack,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 095–14 103.

[30] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, “Defending bit-flip attack through dnn weight reconstruction,” in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[31] Q. Liu, W. Wen, and Y. Wang, “Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2020*, 2020.

[32] M. Yan, C. W. Fletcher, and J. Torrellas, “Cache telepathy: Leveraging shared resource attacks to learn dnn architectures,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2003–2020.

[33] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, “X-deepsca: Cross-device deep learning side channel attack,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[34] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, “Vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design,” in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[35] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research).” [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

[36] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[39] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[40] E. Ozen and A. Orailoglu, “Just say zero: containing critical bit-error propagation in deep neural networks with anomalous feature suppression,” in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.