HammerDodger: A Lightweight Defense Framework against RowHammer Attack on DNNs

Cheng Gongye, Yukui Luo, Xiaolin Xu, Yunsi Fei
Northeastern University, Boston, MA, US
{gongye.c, luo.yuk, x.xu, y.fei}@northeastern.edu

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247671

***Abstract*—RowHammer attacks have become a serious security problem on deep neural networks (DNNs). Some carefully induced bit-flips degrade the prediction accuracy of DNN models to random guesses. This work proposes a lightweight defense framework that detects and mitigates adversarial bit-flip attacks. We employ a dynamic channel-shuffling obfuscation scheme to present moving targets to the attack, and develop a logits-based model integrity monitor with negligible performance loss. The parameters and architecture of DNN models remain unchanged, which ensures lightweight deployment and makes the framework compatible with commodity models. We demonstrate that our framework can protect various DNN models against RowHammer attacks.**

***Index Terms*—Neural networks, Computer security, Fault protection**

I. Introduction

As deep neural network (DNN) models are widely deployed in diverse applications, the attack surface gets expanded from both the algorithm and implementation perspectives. Protecting DNN models against various powerful and stealthy attack methods has become an urgent and challenging task. Recently, bit-flip adversarial weight attacks (BFA) have arisen as a serious security problem on DNN models, which only require the adversary to flip as few as two to twenty bits to corrupt the inference accuracy of a DNN model to the worst-case possible, i.e., random guesses [1], presenting a devastating threat to the run-time execution of DNN models. More severely, it has been revealed that such BFA attacks can be physically launched utilizing the RowHammer phenomenon of DRAM structures, which allows the attacker to accurately flip bits stored in the DRAM from an unprivileged process, as demonstrated in [2]. A recent work [3] discovered that almost all DDR4 DRAMs are vulnerable to RowHammer attacks, making the BFA attack a general and common threat to DNN implementations.

These attacks breach the integrity of DNN models and call for immediate and effective defense solutions, considering the large amount of commodity DNN models deployed in the field. Although some defense solutions against such RowHammer-based BFA attacks have been put forward, their practical deployment presents several challenges, including requiring retraining of the DNN models and demanding significant parameter and structure changes. To address these drawbacks, we propose a defense strategy that makes no changes to the parameters and structure of the original DNN models and does not require any retraining. Such a lightweight defense method removes the barrier that hinders practical defenses against BFA attacks.

This work makes the following contributions.

- We propose a lightweight defense framework, HammerDodger, against RowHammer BFAs on DNNs. To the best of our knowledge, this is the first defense solution that does not require any retraining or significant changes to the model parameters and hyperparameters. It employs a novel *channel shuffling* scheme. It is generally applicable to any commodity DNN models that have convolution layers, dense layers, and batch normalization layers.
- We also propose a run-time model integrity monitoring mechanism. We adopt the distance between the clean logits and logits of the attacked model as a proxy to the inference accuracy degradation, which tracks the accuracy loss with negligible overhead.
- We evaluate the effectiveness of HammerDodger on different datasets including CIFAR-10, CIFAR-100, and ImageNet 2012, as well as across various DNN models. The experimental results demonstrate that our defense method can effectively dodge BFA attacks and maintain the accuracy of the victim DNN models fairly well.

II. Background

A. RowHammer Attack

RowHammer attack has become an emerging threat to applications deployed on platforms with DRAMs [4]. By strategically and frequently accessing the DRAM memory rows around the victim cells, the RowHammer attack can flip the original bit in the victim cells, i.e., 0-to-1 or vice versa. Moreover, a recent work [3] reports improved RowHammer attacks can bypass the Target Row Refresh (TRR) techniques [1] [5], presenting a threat to all the forty DRAM DIMMs from four major vendors tested. As of now, there is no easy fix because TRR is built into DRAM DIMMs and cannot be updated easily, leaving a great number of commodity devices vulnerable to RowHammer attacks.

This work is supported in part by the U.S. National Science Foundation under grants SaTC-1929300, IUCRC-1916762, CNS-2153690, CNS-2247892, and DGE-2043183.

[1]TRR is an umbrella term referring to the various RowHammer mitigation implementations by different vendors in DDR4.

B. Adversarial Bit-flip Weight Attack

Aiming at degrading the performance (e.g., inference accuracy) of a DNN model, the bit-flip-based adversarial weight attack (BFA) leverages the RowHammer phenomenon in the commodity DRAMs. To make BFA attacks effective, the previous works [1], [2], [6] present methods to identify the most vulnerable bits, i.e., these stored in the victim cells, with the largest gradients and lead to the most accuracy loss. BFA can degrade the accuracy of DNN models to random guesses with merely two to twenty bit-flips [1], [2], [6], [7].

Floating-point DNN models' parameters are extremely vulnerable to bit-flip attacks, since even one bit-flip, if located in the exponents, will greatly degrade the accuracy of the DNN model [6], [8]. Hence, floating-point DNN models are not suitable for security-sensitive applications, where quantized models are preferred for their stronger robustness and higher performance. BFA attacks still present a serious and practical threat to quantized models.

C. Defense Solutions Against BFA Attacks

Some works attempt to improve the robustness of DNN models against BFA attacks, e.g., by modifying the model parameters and structures. He et al. proposed to binarize the DNN model weights and retrain the victim model to reduce the effectiveness of BFA attacks [9]. Li et al. added a weight reconstruction step at each layer of the DNN model, to mitigate the weight changes caused by bit-flips [10]. Liu et al. proposed to quantize and retrain the DNN model and change the bit level representation of the weight parameters, to reduce the chance that the attack can successfully target the critical bits [11].

Although effective, deploying these defense solutions is challenging due to their requirement for re-training. Practically the dataset is not always available for the DNN model users. Moreover, training a high-performance DNN model requires massive computational power and domain expertise, which present a barrier to normal DNN model consumers. Even worse, changing the structural and bit-level representation of the DNN models might make them incompatible with various accelerators.

III. Threat Model

Without loss of generality, we adopt the same threat model as the original BFA work [1], [2], [6], which utilizes RowHammer to flip the critical bits of DNN models to degrade their inference accuracy. Fig. 1 illustrates the steps. The victim DNN model's weights are quantized to 8 bits [1], [2]. We assume the strongest attacker, equipped with a great deal of domain knowledge and capabilities, detailed as follows.

1) Victim: As shown in Fig. 1, the victim DNN model is executed in the cloud computing infrastructure with shared DRAM between the victim user and an attacker. The victim model will be loaded into the DRAM (④) for inference (⑤) by a DNN accelerator, e.g., CPU, GPU, or FPGA. The user can get the inference result from the cloud (⑥).

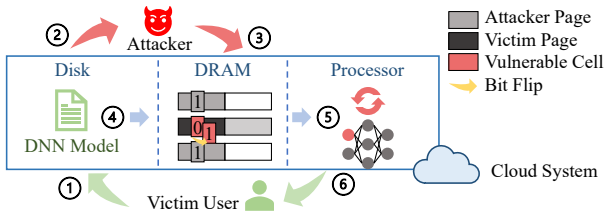


Fig. 1: Illustration of the threat model.

2) Attacker's Knowledge: Following [1], we assume the attacker knows all the parameters and hyperparameters of the victim DNN model [1], [2], [6]. Through reverse-engineering and public research, the attacker also knows the physical layout of the DRAM and the virtual-to-physical address mapping information.

3) Attacker's Capabilities: As Fig. 1 illustrates, the attacker can read the memory using unprivileged technologies (②), such as KPTI bypassing [12]. She will adjust the "hammering" behavior towards vulnerable memory cells. Then, the attackers can profile the victim DNN model and identify the vulnerable bits. Finally, using the advanced memory massage technique (③), the attacker can force the victim page to reside in exploitable vulnerable regions [2].

IV. HammerDodger: A Retraining-free Defense Methodology

We introduce the design rationals of HammerDodger based on three key observations, with several preliminary experiments performed on an 8-bit quantized ResNet-20 model with 92.6% prediction accuracy on the CIFAR-10 dataset.

A. Key Observations

Observation 1: *The possible number of bit-flips, in reality, is not large.*

Although many existing defense works against BFA attacks assume the existence of hundreds of bit-flips that can be produced by the RowHammer attack. We argue this is an impractical assumption since it is challenging to reliably trigger more than one bit-flip in a physical DRAM page of 4 KB. For a modest 8-bit quantized ResNet20 model, the maximum bit-flips possible is $\lfloor \frac{27KB}{4KB} \rfloor = 68$. Moreover, to inject hundreds of bit-flips, the attacker will have to massage hundreds of aggressor rows, which will take a long time, i.e., days or even several weeks [3]. Hence, we argue that it is sufficient to analyze the defense framework under a dozen of bit-flips, which is in accordance with the existing work [2] that assumes an attacker can only flip one bit per page when performing BFA using the RowHammer attack.

Observation 2: *Randomly flipping bits in DNN model weights barely degrades the model prediction accuracy.*

We test the accuracy degradation of a DNN model under two types of attacks, BFA and the random bit-flip attack. We repeat each attack type 20 times with different random seeds. The testing results are presented in Fig. 2, which shows that the BFA can significantly degrade the model prediction

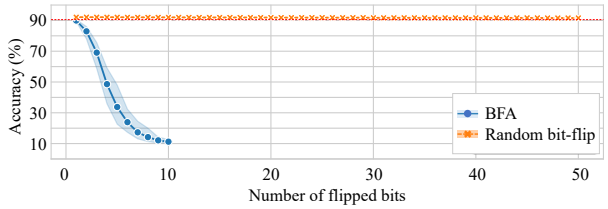


Fig. 2: Model prediction accuracy (with a 95% error band) vs. the number of flipped bits under two attack strategies.

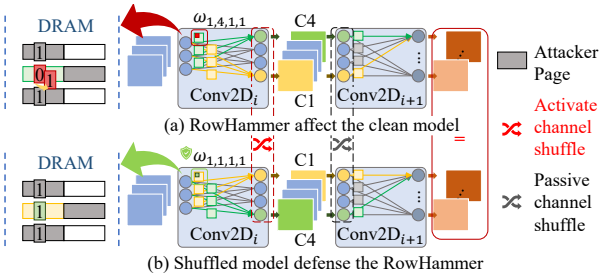


Fig. 3: Channel-wise DNN model shuffling. (a) The clean model inference process between two convolutional layers. (b) Apply a synchronized model shuffling sequence along the output channels of Conv2D$_i$ and the input channels of Conv2D$_{i+1}$, respectively.

accuracy, i.e., to random guesses (10% in CIFAR-10) with only 10 flips. On the contrary, the random bit-flip attack barely affects the prediction accuracy, even with more bit-flips. These experimental results demonstrate that most DNN model weights are naturally resilient to such bit-flip attacks, in terms of model inference accuracy. This observation motivates the defense strategy: *degrading the BFA to a random bit-flip attack can effectively mitigate the RowHammer-based BFA.*

Observation 3: *The channels of a DNN model are independent, and shuffling along the channel dimension does not affect the final result.*

We illustrate this key observation in Fig. 3 using two consecutive convolutional layers, $Conv2D_i$ and $Conv2D_{i+1}$, where $Conv2D_i$ has 3 input channels and 4 output channels, and $Conv2D_{i+1}$ has 4 input channels and 2 output channels. Assuming the BFA attack aims to flip a critical bit in the fourth output channel of $Conv2D_i$ (the green C4 channel). To dodge the attack, an *active channel shuffle* along the output channel of $Conv2D_i$ can move the critical kernels away, e.g., Channel 1 (yellow) and Channel 4 (green) are swapped. Note that since shuffling the output channels of a layer will also shuffle the output feature maps, the corresponding input channel kernels in the next adjacent layer, $Conv2D_{i+1}$, should also be adjusted to align with the previous layer's output channel shuffling, defined as a *passive channel shuffling*. Due to the channel independency, such alignment guarantees the correctness of the output. For a convolutional layer with n output channels, its shuffling space is $n!$, which can be huge especially for larger DNN models with dozens and even hundreds of channels.

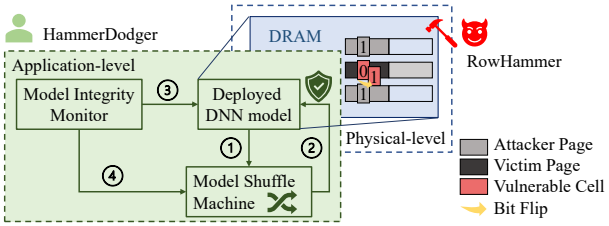


Fig. 4: HammerDodger Overview.

B. HammerDodger: System Overview

From the first two observations, we conclude that if we can frequently change the physical location of the most vulnerable model bits in the DRAMs, we can essentially degrade the impact of BFA attacks, i.e., to random bit-flip attacks. Practically, we do not need to make any fine-grained changes to the DNN model weights, i.e., at the bit-level, but can achieve this by shuffling the channels as demonstrated in **Observation 3**.

To launch this defense strategy, we propose two building blocks, as shown in Fig. 4. The first one is the model shuffle engine, which periodically executes the channel-wise model shuffling (① → ②), to move the target of the BFA attacks. The second building block is a model integrity monitor, which checks the accuracy degradation (③) caused by the random bit-flips at run-time and reloads the model from the disk at a certain degradation threshold, so as to recover the model prediction accuracy (④ → ②). We present details of these building blocks in the flowing subsections.

C. Model Shuffle Block

The model shuffle block consists of two parts. The first part is a shuffle-enabled model loader, which aims to avoid the original model from being loaded into the memory because the attacker may have RowHammer traps [2] that can quickly deploy the BFA attack [13]. We implement this part using the *memap* function of Numpy [14], which allows us to map the model file without actually loading it into the memory. Then, we can load the model weights using the shuffled sequence from the memory map. The second part is the online model shuffler, which shuffles the model on demand after the model is stored in the memory. Both parts use the same shuffling algorithm, which supports permuting generic DNN layers, including the convolution layer, batch normalization layer, dense layer, and residue blocks. Therefore, the model shuffle block is compatible with any popular DNN architectures like VGG [15], AlexNet [16], ResNet [17], and MobileNet [18]. Since our framework only introduces channel-wise shuffling as described in **Observation 3**, we do not need to tackle those special layers like activation layers and pooling layers which do not have parameters to relocate. Cognizant of the output channels, the input channels of the next layer need to

[2]Note there is no work that implements this kind of traps for DNN yet. We consider the worst possible scenario to build strong defense, where the attacker employs memory exhaustion techniques to force the DNN model to locate at the vulnerable DRAM region upon initial loading.

Algorithm 1: Algorithm to shuffle the weight of two consecutive layers

Input: Weights of Layer i and $i+1$: $\boldsymbol{w_i}$ and $\boldsymbol{w_{i+1}}$ of dimension $[IC, OC, W, H]$

Output: shuffled weights $\boldsymbol{w_i}'$ and $\boldsymbol{w_{i+1}}'$

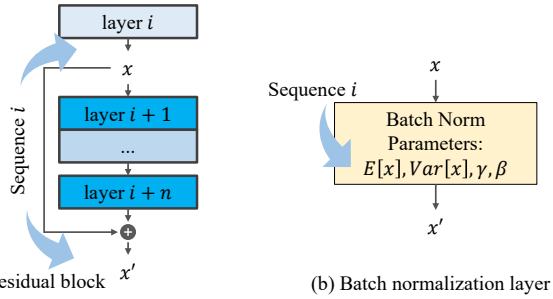
$Index = 1$

for *RandIndex in* $\boldsymbol{s}$ **do**

  $\boldsymbol{w_i}'[:, Index, :, :] = \boldsymbol{w_i}[:, RandIndex, :, :]$

  $\boldsymbol{w_{i+1}}'[Index, :, :, :] = \boldsymbol{w_{i+1}}[RandIndex, :, :, :]$

  $Index = Index + 1$



(a) Residual block (b) Batch normalization layer

Fig. 5: Shuffle examples. (a) Shuffle sequence distribution with complex links, such as residual block. (b) The shuffle sequence should be applied to all channel-wise parameters.

be shuffled accordingly. We demonstrate how to shuffle two consecutive convolution layers in Alg. 1. Note that the number of output channels (OC) of layer i is always the same as the number of the input channels (IC) of layer $i+1$, i.e., $OC_i \equiv IC_{i+1}$. Therefore, we generate a random channel sequence $\boldsymbol{s}$, ranging from one to OC_i, and use this sequence to direct the memory layout of the output channel weights of layer i and the input channel weights of layer $i+1$.

HammerDodger keeps the input channel of the first layer and the output channel of the last layer unchanged, and shuffles both the input and output channels of hidden layers. For each layer i, we generate a random sequence s_i of length OC_i. Two situations need special attention: residue blocks and batch normalization layers. We demonstrate how to shuffle a residue block in Fig. 5(a). A residue block has an extra connection compared with regular DNN layers. The output $\boldsymbol{x}$ of layer i not only goes through layer $i+1$ but also adds to the output of layer $i+n$ to generate $\boldsymbol{x}'$. Hence, the output channel of layer i and layer $i+n$ must coordinate to follow the same shuffle sequence. The batch normalization layer has various values for its four parameters: running mean ($E[\boldsymbol{x}]$), running variance ($Var[\boldsymbol{x}]$), weight(γ), and bias(β) in each channel. As shown in Fig. 5(b), we have to shuffle all four parameters according to the output sequence i of the previous layer.

In addition to the shuffling mechanism, we should determine the shuffling frequency based on the capability of the attacker. Since our defense goal is to degrade the BFA attack to a random bit-flip attack, we need to shuffle the DNN model fast enough so that the attacker cannot effectively perform the surgical BFA attacks.

Algorithm 2: Model integrity monitoring algorithm.

Input: DNN Model M, input $\boldsymbol{I_i}$, clean logits $\boldsymbol{L_i} = M(\boldsymbol{I_i}), i \in (1, n)$, the reload threshold th_L

$D_i = 0, i \in (1, n)$

$i = 1$

while *True* **do**

  if M *inferences one batch of inputs* **then**

    $\boldsymbol{L_i}' = M(\boldsymbol{I_i})$

    $D_i = \frac{\|\boldsymbol{L_i'} - \boldsymbol{L_i}\|_1}{N}$

    if *Average*$(D_1 ... D_n) > th_L$ **then**

      $ReloadModel$

      $D_i = 0, i \in (1, n)$

    $i = i \mod n$

    $i = i + 1$

To determine the shuffling frequency, we assume a strong attacker, who can locate the target vulnerable bits in the DRAM instantly, i.e., after shuffling. Then, the attacker needs to massage the aggressor rows, so they can flip the vulnerable bits. From the defense perspective, if we shuffle the DNN model channels before the attacker can finish the massaging, we can degrade the BFA to random bit-flip attacks. We consider the state-of-the-art memory massaging technique that takes dozens of seconds to flip a target row [2]. Hence, shuffling every second should effectively mitigate the BFA. We present the accuracy and overhead analysis in Sec. V.

D. Model Integrity Monitor

When a victim model is under a BFA, the accuracy would degrade drastically as the number of bit-flips increases, as shown in Fig. 6. Although the average accuracy is still high when our channel shuffling defense is applied, we observe the distance between the min and max inference accuracy is large. This is because as the number of trials increases, there is still a chance that the random bit-flips fall on the vulnerable bits, to introduce great performance degradation. To address this potential issue, we introduce a model integrity monitor block, which is used to track the accuracy degradation and reload the model when needed.

The most accurate way to measure the model's accuracy degradation is to run the whole validation dataset, which is however, too costly since the validation dataset typically has thousands of images. To achieve lightweight validation, i.e., model integrity monitoring, we develop a novel proxy that can reflect the accuracy degradation using only 20 inferences. Specifically, we use the distance between the logits of the clean model and the logits of the post-attack model of the same input image as the metric, to reflect the accuracy degradation. Generally, the larger the distance, the more bits in the victim model are flipped, corresponding to a higher accuracy degradation. During the offline stage, we characterize the relationship

TABLE I: Performance Overhead of HammerDodger

Dataset	Network	Number of classes	Input size	Number of training samples	Number of validation samples	Number of parameters (M)	Shuffle time (ms)	Reload time (ms)	Monitor time (ms)	Inference throughput (fps)	
										Original	HammerDodger w/o reloading
CIFAR-10	ResNet-20	10	32×32	50K	10K	0.27	2.44	25.71	0.42	2370	2351
CIFAR-100	ResNet-32	100	32×32	50K	10K	0.47	4.71	37.3	0.61	1641	1628
ImageNet	ResNet-18	1,000	224×224	1.2M	50K	11.69	14.36	209.71	1.63	86.3	85.1

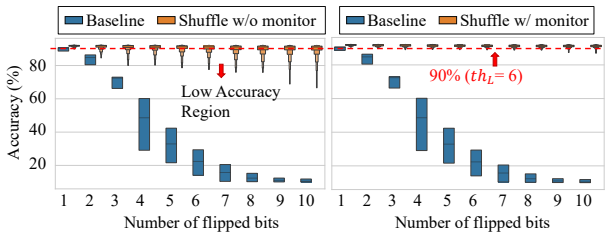


Fig. 6: Model shuffle performance w/o & w/ model integrity monitor.

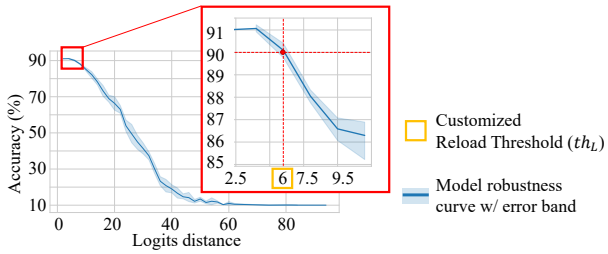


Fig. 7: Model robustness curve example, where we apply the target ResNet18 model w/ reference image set.

between the accuracy degradation and the logits' distance, and build a look-up table for the online inference to check. We use the average of 20 distances to increase the stability of the measurement. In Fig. 7, we show an example offline profiling step using ResNet20 and the CIFAR-10 dataset, from which we can see that the logits-distance provides a high-fidelity estimation of the model accuracy.

We describe this proxy-enabled model accuracy monitoring in Alg. 2. The input of the algorithm is the 20 randomly selected input images and their corresponding clean logits. The user selects a *model-reload threshold* based on the offline profiling result and their desired model accuracy. The accuracy monitoring program runs inference on one of the 20 images and calculates the distance between the current logtis and the clean logits. If the average value of the distance is greater than the threshold, the monitoring program reloads the model from the disk to obtain a clean model. Note that if there is no attack or the accuracy degradation is acceptable, the model will not be reloaded, minimizing the performance overhead.

V. Evaluations

A. Experimental Setup

Dataset and DNN Architectures: To thoroughly evaluate the performance of HammerDodger, we adopt three popular classification datasets that span small, medium, and large scales: CIFAR-10 [19], CIFAR-100 [19], and ImageNet 2012 [20]. We list the metadata of the dataset in Table I. Moreover, we select the model architectures based on the publicly available pre-trained models from reputable sources. The pre-trained parameters of ResNet18 [17] for ImageNet are from the official PyTorch Model Zoo [21]. The parameters of ResNet20 and ResNet32 are from the public model zoo of Chen et al. [22], whose training codes and logs are open. Note that we choose the configurations of ResNet tailored for CIFAR and ImageNet respectively according to the original ResNet paper [17]. The ResNet18 is much larger than the ResNet20 and ResNet32.

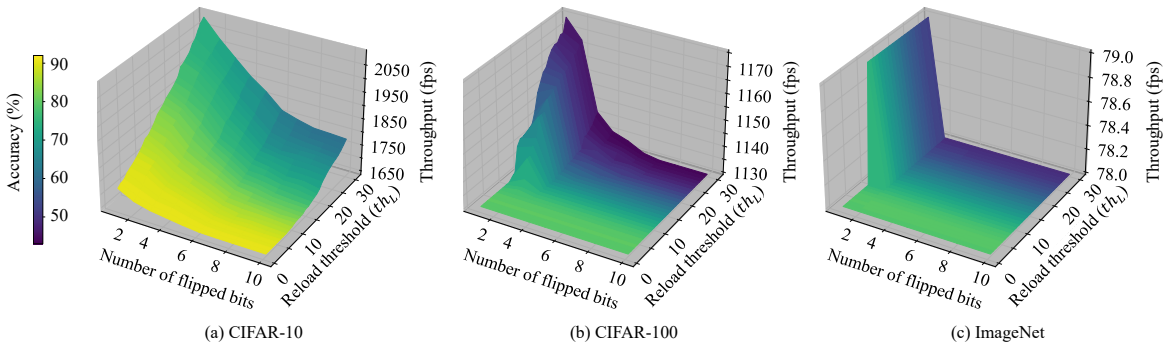
BFA Configuration: We apply the same configuration for vulnerable bits searching as the prior work [1]. Specially, we relax the restriction that only allows flipping one bit per 4 KB page, to mimic the capabilities of the strongest possible attacker. For each dataset and network configuration, we generate 20 sets of bit-flips that can degrade the inference accuracy of the DNN model to random guesses using different random seeds, to ensure coverage of the testing. On the defense evaluation side, we generate $4,000$ shuffling permutations for each network to produce stable numeric results.

Experiment Platform: We perform the BFA experiment and accuracy evaluation using NVIDIA TITAN X (Pascal) GPU with PyTorch 1.12.1 and CUDA toolkit 11.3.1. The overhead evaluation runs on a workstation with Intel Xeon CPU E5-2667 v4 @ 3.20 GHz CPU, 4x8 GB Hynix DDR4 memory @ 2400 MHz, and 2 TB Samsung EVO 870 SSD.

B. Performance Evaluation

In this section, we evaluate the overhead of HammerDodger and analyze the suitable configuration that provides a good trade-off between performance and defense strength. We present the overhead of HammerDodger in Table I, where we show the average time of $1,000$ trials to perform shuffle, monitor, and reload during one model inference, respectively. The inference time is the overall run time of one batch (128 images). Since we only need to shuffle the model once a few seconds, the shuffling overhead is small compared to the inference time. Therefore, the shuffling will not significantly affect the throughput of the DNN model. The model reloading, however, is relatively expensive since it restores all the parameters of the DNN model. The performance shown in Table I Column 11 and 12 are the baselines (without reloading).

In Fig. 8, we show the accuracy and performance trade-off when reloading the model using different thresholds. First, we can see that HammerDodger can mitigate RowHammer-based BFA effectively. When we set the reloading threshold to less



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Fig. 8: Trade-off analysis between model throughput and accuracy.

than 6, all three models can maintain the original accuracy with trivial to no loss. Second, the medium and large size CIFAR-100 and ImageNet models are more sensitive to faults, i.e., the probability of reloading quickly becomes high, and the throughput plateaus when there are merely one to three bit-flips. The good news is that the throughput degradation is less than 10% for ImageNet, since for large models, the reloading time is short in relevance to the inference time. Hence, for large datasets and models, in practice, we can select a small reload threshold to effectively protect the DNN model at a reasonable throughput degradation. For smaller dataset and models like CIFAR-10, CIFAR-100 the performance penalty is higher because the reload time occupies a more significant portion than the other two cases. The user can decide the trade-off between accuracy and performance. For example, a user can choose a higher reloading threshold at the expense of slightly worse accuracy to improve the throughput.

VI. Discussion and Conclusion

This paper presents a lightweight defense framework, HammerDodger, against the powerful and stealthy bit-flip attacks using RowHammer on DRAMs. HammerDodger can protect DNN models with nearly no accuracy loss, using dynamic model shuffling to degrade the BFA attacks to random bit-flip attacks. HammerDodger also monitors the accuracy of the DNN model using a novel logits-based distance measurement and restores the model when necessary. We present detailed performance overhead analysis on various datasets and network structures. In terms of defense effectiveness, HammerDodger is as effective as the strongest previous solution [9]. The largest advantage of HammerDodger lies in the removal of DNN model re-training, i.e., both parameters and structure, presenting a user-friendly and efficient defense technique. Built on PyTorch, HammerDodger can be applied for protecting any commodity DNN models that have convolution layers, dense layers, and batch normalization layers, and is compatible with most DNN accelerators.

References

[1] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, 2019.

[2] F. Yao, A. S. Rakin, and D. Fan, "{DeepHammer}: Depleting the intelligence of deep neural networks through targeted chain of bit-flips," in *USENIX Security*, 2020.

[3] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *S&P*, 2022.

[4] 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*, 2014.

[5] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *S&P*. IEEE, 2020.

[6] 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 *USENIX Security*, 2019.

[7] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant fpga," in *USENIX Security*, 2021.

[8] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga," in *DAC*. IEEE, 2021.

[9] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in *CVPR*, 2020.

[10] 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 *DAC*. IEEE, 2020.

[11] L. Liu, Y. Guo, Y. Cheng, Y. Zhang, and J. Yang, "Generating robust dnn with resistance to bit-flip based adversarial weight attack," *IEEE Transactions on Computers*, 2022.

[12] A. Luțaș and D. Luțaș, "Bypassing kpti using the speculative behavior of the swapgs instruction," in *BlackHat*, 2019.

[13] 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 *S&P*. IEEE, 2018.

[14] "numpy.memmap NumPy v1.23 Manual." [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.memmap.html

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

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[19] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, 2015.

[21] "Models and pre-trained weights Torchvision 0.14 documentation," Nov. 2022. [Online]. Available: https://pytorch.org/vision/stable/models.html

[22] chenyaofo, "pytorch-cifar-models," Nov. 2022. [Online]. Available: https://github.com/chenyaofo/pytorch-cifar-models