DNN-Defender: A Victim-Focused In-DRAM Defense Mechanism for Taming Adversarial Weight Attack on DNNs

Ranyang Zhou[†,*], Sabbir Ahmed[‡,*], Adnan Siraj Rakin[‡], and Shaahin Angizi[†]
[†]Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
[‡]Department of Computer Science, State University of New York at Binghamton, NY, USA
[*]These authors contributed equally
rz26@njit.edu,sahmed9@binghamton.edu,arakin@binghamton.edu,shaahin.angizi@njit.edu

ABSTRACT

With deep learning deployed in many security-sensitive areas, machine learning security is becoming progressively important. Recent studies demonstrate attackers can exploit system-level techniques exploiting the RowHammer vulnerability of DRAM to deterministically and precisely flip bits in Deep Neural Networks (DNN) model weights to affect inference accuracy. The existing defense mechanisms are software-based, such as weight reconstruction requiring expensive training overhead or performance degradation. On the other hand, generic hardware-based victim-/aggressor-focused mechanisms impose expensive hardware overheads and preserve the spatial connection between victim and aggressor rows. In this paper, we present the first DRAM-based victim-focused defense mechanism tailored for quantized DNNs, named *DNN-Defender* that leverages the potential of in-DRAM swapping to withstand the targeted bit-flip attacks with a priority protection mechanism. Our results indicate that DNN-Defender can deliver a high level of protection downgrading the performance of targeted RowHammer attacks to a random attack level. In addition, the proposed defense has no accuracy drop on CIFAR-10 and ImageNet datasets without requiring any software training or incurring hardware overhead.

CCS CONCEPTS

• **Hardware → Dynamic memory**; • **Security and privacy → Security in hardware**.

ACM Reference Format:
Ranyang Zhou, Sabbir Ahmed, Adnan Siraj Rakin, and Shaahin Angizi. 2024. DNN-Defender: A Victim-Focused In-DRAM Defense Mechanism for Taming Adversarial Weight Attack on DNNs. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3656222

1 INTRODUCTION

The far-reaching development of Deep Neural Network (DNN) accuracy even with low-bit-width models has recently triggered various security-associated attacks in many applications [13]. Recent studies show that an adversary can identify and manipulate a small number of vulnerable bits of off-the-shelf well-trained DNN weight parameters to significantly compromise the output accuracy [4, 13]. Such Bit-Flip Attacks (BFAs) have been enabled mainly due to a manifestation of a DRAM cell-to-cell interference and failure mechanism called RowHammer (RH) [7]. RH attack is conducted when a malicious process activates and pre-charges a specific row (i.e., aggressor row) repeatedly to a certain threshold (T_{RH}) to induce bit-flips on immediate nearby rows (i.e., victim rows). Unfortunately, by scaling down the size of DRAM chips in the modern manufacturing process, DRAM becomes increasingly more vulnerable to RH bit-flips [6]. Figure 1(a) shows that the RH threshold has had a significant downward trend in recent years, e.g., the attacker needs ~4.5× fewer hammer counts on LPDDR4 (new) as opposed to DDR3 (new) [21].

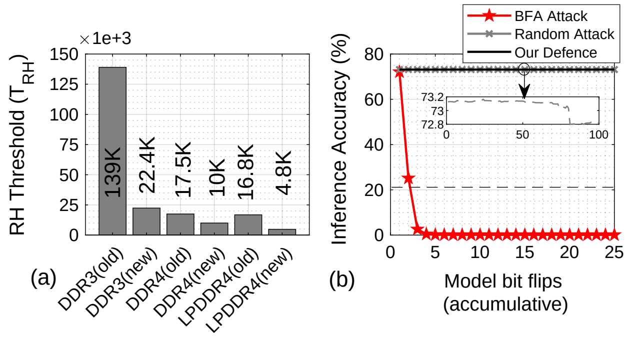

Figure 1: (a) RowHammer thresholds [21], (b) Targeted vs. random bit flipping for an 8-bit ResNet-34 on ImageNet and *DNN-Defender*'s performance.

To prevent RH attacks, DRAM manufacturers and researchers have proposed hardware-based victim-focused defense mechanisms to proactively refresh the victim rows by adding counters to count the number of activations [7]. However, such RH mitigation proposals have faced a huge overhead both from latency and power consumption perspectives [24]. To mitigate this, recent aggressor-focused swap-based mechanisms [16, 21] proactively swap and unswap aggressors with random rows before reaching the RH threshold. Such mechanisms can be immensely effective when the attacker does not have knowledge of the internal DRAM organization. The Randomized Row-Swap (RRS) [16] swaps the aggressor row with a random row within the same bank in the memory. A method called Secure Row-Swap (SRS) [21] has demonstrated the use of fewer counters for crucial data and implemented associated threat mitigation using the swap operation. This approach reduces the swap rate while maintaining security, resulting in higher efficiency and lower latency. Assuming the attacker's access to this information, the attacker will not track the aggressor row but the victim row and attack its adjacent row, making it a new aggressor row. In this case, swapping the aggressor row with another

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3656222

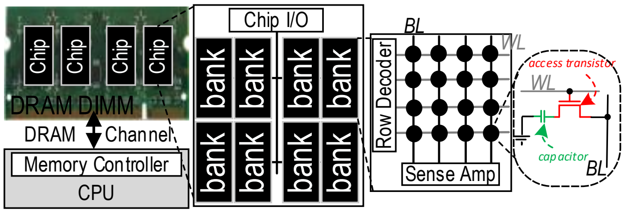

Figure 2: Organization of a DRAM chip.

random row is purposeless. On the other side, prior works have approached the problem of DNN weight noise mitigation from a software-training optimization perspective [3, 9] and DNN architecture modification [14] that impose expensive overhead or performance/accuracy degradation.

We develop DNN-Defender as a pure DRAM-based victim-focused defense mechanism to effectively withstand the targeted RH BFAs on DNNs. The main contributions of this work are: (1) We design the DNN-Defender mechanism with hardware-software support that utilizes in-DRAM swapping to protect the DNN weight parameters not requiring any software training or imposing additional hardware overhead; (2) We develop a priority protection mechanism method and parallelism to tailor the performance-accuracy trade-offs with respect to the system requirements; and (3) We extensively analyze the DNN-Defender's applicability and efficiency in taming RH vulnerability compared to recent hardware/software techniques over CIFAR-10 and ImageNet DNN datasets. On the CIFAR-10, the accuracy of the DNN-Defender-supported system is 91.71% while the baseline indicates only 10.9% under BFA.

2 BACKGROUND & MOTIVATION

2.1 DRAM

Organization & Commands. The DRAM chip is a hierarchical structure consisting of several memory banks as shown in Fig. 2. Each bank comprises 2D sub-arrays of memory bit-cells that are virtually ordered in memory matrices (mats), which have billions of DRAM cells on modern chips. Each DRAM bit-cell consists of a capacitor and an access transistor. The charge status of the bit cell's capacitor is used to represent binary "1" or "0" [17]. In idle mode, the memory controller turns off all enabled DRAM rows by sending the Precharge (PRE) command on the command bus. This will precharge the Bit-Line (BL) voltage to $\frac{V_{DD}}{2}$. In the active mode, the memory controller will send an Activate (ACT) command to the DRAM module to activate the Word-Line (WL). Then, all DRAM cells connected to the WL share their charges with the corresponding BL. Through this process, BL voltage deviates from the precharged $\frac{V_{DD}}{2}$. The sense amplifier then senses this deviation and amplifies it to V_{DD} or 0 in the row buffer. The memory controller can then send read (RD)/write (WR) commands to transfer data to/from the sense amplifier array.

RowClone. Exploiting the fact that DRAM transfers an entire row of data to the corresponding row buffer during the read operation, RowClone [18] has been developed as a simple and efficient mechanism to enable a bulk in-memory copy operation (<100ns) from a source row to a destination row completely in the DRAM sub-array. RowClone eliminates the need to transfer data over the memory channel. The memory controller manages this by issuing two back-to-back ACT commands first to the source and then the destination without a PRE command in between with almost negligible cost. By using this method, the latency and power consumption of a bulk copy operation can be reduced by a factor of 11.6 and 74.4, respectively [18].

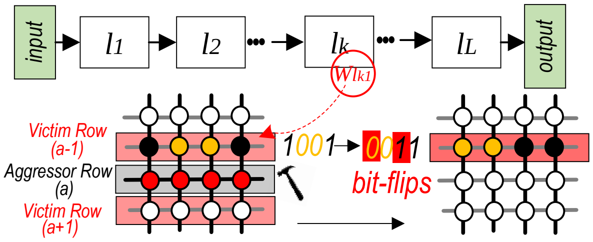

Figure 3: Adversarial weight RowHammer attack in the k_{th} layer of a DNN.

2.2 Row Hammer-based DNN Weight Attack

The BFA progressively searches for vulnerable bits by first performing a bit ranking within each layer based on gradient [13]. Considering a weight quantized DNN, the weight matrix can be parameterized by two complement representations $\{\mathbf{B}_l\}_{l=1}^{L}$, where $l \in \{1, 2, ..., L\}$ is the layer index. BFA computes the gradient w.r.t. each bit of the model ($|\nabla_{\mathbf{B}_l}\mathcal{L}|$) where $\mathcal{L}$ is the inference loss function. At each iteration, the attacker performs two key attack steps: i) inter-layer search and ii) intra-layer search; where the goal is to identify a vulnerable weight bit and flip it. Given a sample input x and label t, the BFA [13] algorithm tries to maximize the following loss function ($\mathcal{L}$):

$$\max_{\{\hat{\mathbf{B}}_l\}} \mathcal{L}\Big(f\big(\boldsymbol{x};\{\hat{\mathbf{B}}_l\}_{l=1}^{L}\big),\boldsymbol{t}\Big), \qquad (1)$$

while ensuring the hamming distance between the perturbed weight tensor by BFA ($\hat{\mathbf{B}}_{l=1}^{L}$) and initial weight tensor ($\{\mathbf{B}_l\}_{l=1}^{L}$) remains minimum. Finally, the attack efficiency can be measured by the number of bit-flips required to cause DNN malfunction. Figure 1(b) illustrates how the DNN accuracy degrades under a few (less than 5) targeted bit-flips using DeepHammer attack [23] in an 8-bit quantized ResNet-34 running the ImageNet dataset as opposed to over 100 random BFAs. Figure 3 illustrates how such an adversarial weight attack is conducted in an L-layer DNN on the target $W_{l_{k1}}$ = 1001, i.e., the weight located in the l_k layer. The malicious process continuously hammers the aggressor row in a-address and induces bit-flips on adjacent victim rows ($a-1$ and $a+1$) holding $W_{l_{k1}}$. As a result, such a single-sided RH attack changes the weight value (herein, 1001→$\underline{0}0\underline{1}1$).

3 WHITE BOX THREAT MODELS

Hardware Threat Model. We assume the following threat model: 1) Each row has a threshold T_{RH} after becoming an aggressor row, and once exceeded within the refresh interval (T_{ref}), it will impose a bit-flip to two adjacent victim rows; 2) We assume that all vulnerable data rows are neither concentrated in one/two sub-arrays nor evenly distributed in each sub-array. Experimentally, most sub-arrays store several data rows simultaneously; some may store multiple or none; and 3) The attacker has a detailed mapping file as shown in Fig. 4 that can locate the physical address of the target data in the neural network and is aware of the initial static mapping of the DRAM rows (i.e., physical adjacency information between rows) [5, 20]. Therefore, the attacker can perform an RH attack on the targeted content.

Table 1: Standard threat model for BFA [13, 23].

Information	Attackers Access
Model architecture and parameters	✓
Small batch (e.g., 128) of test data	✓
Address of parameter cached in DRAM	✓
Model training data and configuration	×
Memory read & write permission	×

Software Threat Model. We assume a standard white-box threat model for the BFA adopted across multiple attack domains in prior works [13, 15]. In a white-box threat model (summarized in Table 1), the attacker is aware of the internal structure of the DNN models, e.g., the number of layers and the width of each layer. On top of that, the attacker has complete knowledge of the DNN model parameters, their values, and bit representation for inference. This assumption is practical due to the recent advancement of side-channel information leakage and the reverse-engineering of DNN models [22] that can recover the DNN model configuration at the inference stage. However, the attacker can not access the training stage configuration and data even in a white-box setting. The attacker has a sample batch of test data to launch the attack. Moreover, the attacker is co-located with the victim, which enables the attacker to run user-space unprivileged processes. Finally, the attacker has complete knowledge of the DRAM addressing scheme. In this work, two types of white-box attack threat models are considered. A *semi-white-box attack*, as a weak version of the BFA, where the attacker is unaware of our proposed defense scheme, and a *complete white-box attack*, where the attacker is aware of the defense and takes necessary actions to circumvent the mitigation.

4 DNN-DEFENDER MECHANISM

The DNN-Defender mechanism is developed to utilize the minimum number of swap operations and to optimize latency overheads to protect the DRAM against the targeted RH in the adversarial weight attack. Our mechanism makes traceability very difficult for the attacker. Even though the attacker can precisely locate the target data, the DNN-Defender only requires to perform multiple swap operations to the victim rows to secure the memory. The memory instruction-based swap allows us to avoid concerns regarding invalid refreshes resulting from a communication delay between the counter and the counter table and off-chip access as seen in previous designs [8, 11, 16, 19, 21]. Besides, our design prioritizes versatility over assessing the potential impact of swap operations on data in different scenarios and determining the worst-case scenario. The focus is on preventing all threats regardless of the circumstances. However, since an aggressor row is typically accompanied by more than one victim row, one can inevitably ignore the other victim rows when focusing on the target data since the impact of such data on the final result is far less than that of important data. However, to minimize the adverse effect, DNN-Defender also uses the lowest resources possible to ensure their security.

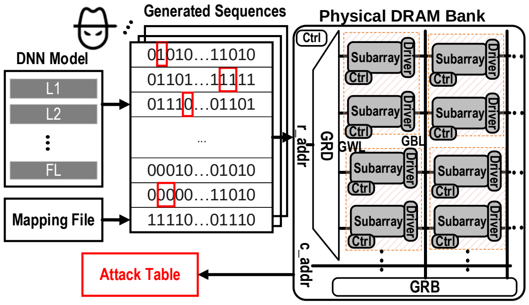

Figure 4: Threat model about attacker's known information about DNN and the attack process.

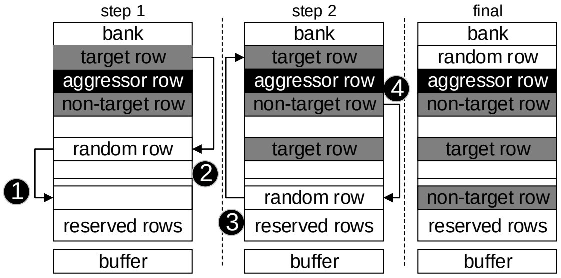

Figure 5: The four-step swap mechanism in DNN-Defender.

As shown in Fig. 5, we propose to virtually partition the protected data region in each memory sub-array into the target and non-target victim rows according to their protection priority. The target row holding the targeted DNN weight is the row with the highest priority to be protected. In other words, if a bit(s) within a target row is flipped, the final DNN accuracy will significantly drop. The non-target rows however show a certain degree of tolerance to errors as they have no/negligible effect on the final result as illustrated in Fig. 1(b). However, in extreme continuous attack scenarios, it is not ruled out that non-target rows may contain important data that would adversely reduce the DNN accuracy. Therefore, as illustrated in Algorithm 1, DNN-Defender not only needs to guarantee the security of the target row but also furnishes a low-cost safeguard for the non-target row. The DNN-Defender's swap operations are accomplished in four steps. As depicted in Fig. 5, in step ❶, the memory controller selects a random row in the sub-array and leverages RowClone [18] to copy it to the reserved row. In step ❷, the target row is copied to the random row in the same way. This in-memory operation will refresh the target row and reset the attacker's target. The rationale is that even though the target row is copied to another position, the malicious process knows the new location, so it will move to the latest row beside the swapped target row and make it a new aggressor row. In step ❸, the random row in the reserved rows region is copied back to the original location of the target row. Aiming to refresh the non-target row knowing that the capacity of the reserved rows region is limited, in step ❹, the non-target row is copied to the reserved row until the next random row overwrites it. Finally, the original target row and random row are swapped, and the non-target row is refreshed. Please note that the attacker can track the target rows and attack the updated addresses, while they will no longer attack the non-target and random rows. It is worth pointing out unlike SRS [21] and RRS [16], which focus on the aggressor row, DNN-Defender focuses on protecting the victim row.

Timing Considerations. Considering T_{RH} is set to 4,800 in LPDDR4 [21], the victim rows must be refreshed before the activation number reaches the threshold. Therefore, it is necessary for DNN-Defender to complete swapping operations within the threshold window (<4800)×T_{ACT}. DNN-Defender only needs one random row generation to support all swap operations. Figure 6 illustrates the DNN-Defender's defense timeline and parallelism for a sample DNN with multiple swap operations. In swap 1, it requires

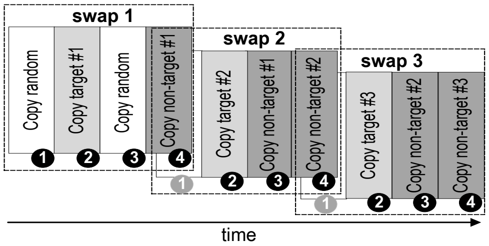

Figure 6: Timeline of a sample DNN with multiple swap ops.

a Random Number Generator (RNG) to define the initial random row for step ❶. The remaining three steps then follow that as we discussed before in Fig. 5. After completing step ❹, DNN-Defender stores the non-target row data of swap 1 in the reserved row, which means that the non-target row 1 at this time can be used as a new random row. Therefore, as shown in Fig. 6, step ❶ of swap 2 can overlap with step ❹ of swap 1. Then it finishes the remaining three steps (❷ to ❹). This method is readily applied to the next swaps.

Priority Protection Mechanism. To select the target rows requiring high-priority protection from DNN-Defender, we propose using the same attack searching algorithm adopted by an attacker for BFA [13]. We use a copy of the victim model with identical model architecture and weights to compute the gradient of the loss function w.r.t. each weight bit ($|\nabla_{\mathbf{B}_l}\mathcal{L}|$). Similar to the BFA attack, we rank the gradients and only flip the highest gradient bit of a specific layer which causes the largest increase in loss function as shown in Eqn. 1. We run the software search algorithms until the model accuracy drops close to the random guess level (e.g., 10 % for CIFAR-10). After performing one complete round of BFA, we record the target bit location $R_1 \in (l, k)$ (l is the layer number, k is the index at a specific layer) that was flipped in the current attack round. Next, we flip back all the targeted bits and perform another round of BFA but this time skip flipping the bits from the previous round R_1. Hence for round R_2, if we encounter any bits from the previous round R_1, we skip it and select the next bit candidate based on gradient. In this way, we keep performing the bit-search algorithm for multiple rounds ($R_c, c = 1, 2, ..., r$), where each round skips all the bits from the previous rounds of the attack. The total number of rounds r depends on the number of bits the defender wants to secure, and increasing r will increase the level of protection. After profiling all the vulnerable bits to BFA for multiple rounds, we select these bit sets as the priority bits requiring more protection. DNN-Defender will prioritize protecting these vulnerable bit sets by selecting them as the target row in their corresponding sub-array.

Algorithm 1 DNN-Defender's Swap Algorithm

```
Procedure: Protection
If DD_Start
    $Define\ random\_row = rand(DRAM)$
    $reserved\_row \leftarrow random\_row$
    $random\_row \leftarrow targe\_row[row]$
    $targe\_row[row] \leftarrow reserved\_row$
        If $(Target\_rows == 1)$ break;
        else For $row$ in $Target\_rows$ do
            $reserved\_row \leftarrow non\_target\_row[row]$;
            $non\_target\_row[row] \leftarrow targe\_row[row + 1]$;
            $targe\_row[row + 1] \leftarrow non\_target\_row[row]$;
else if DD_Interrup
    break;
else continue();
end Procedure
```

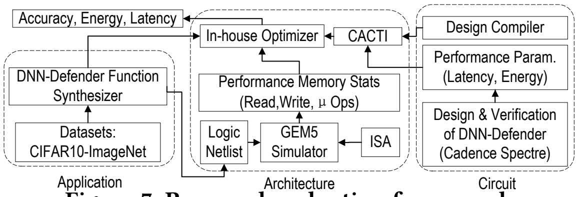
Accuracy, Energy, Latency
DNN-Defender Function Synthesizer
Datasets: CIFAR10-ImageNet
Application
In-house Optimizer
CACTI
Performance Memory Stats (Read,Write, μ Ops)
Logic Netlist
GEM5 Simulator
ISA
Architecture
Design Compiler
Performance Param. (Latency, Energy)
Design & Verification of DNN-Defender (Cadence Spectre)
Circuit

Figure 7: Proposed evaluation framework.

5 EXPERIMENTAL RESULTS

Setup. We present a cross-layer evaluation framework as shown in Fig. 7 to demonstrate the benefits of DNN-Defender. Firstly, we developed DNN-Defender's sub-arrays with peripherals using Cadence Spectre in the 45nm NCSU PDK library [1] at the circuit-level to verify functionality, attain performance parameters, and measure the row-shuffle time. The memory controller and registers were designed and synthesized by Design Compiler with a 45nm industry library. Afterward, we incorporated the results from circuit-level assessments and extensively modified CACTI at the architecture-level. We implemented DNN-Defender's ISA in gem5 [2], and exported the memory statistics and performance to an in-house C++ DNN-Defender optimizer, taking the CACTI output and application netlist as the inputs. At the application, we evaluated the performance of our proposed technique in defending against adversarial BFA using two commonly-used visual datasets: CIFAR-10 and ImageNet. The weights were quantized to 8-bit width. To carry out the BFA, we randomly sampled images from the test/validation set, with a default sample size of 128 for both datasets.

5.1 Performance Evaluation

Hardware Overhead Analysis. We compare the DNN-Defender's hardware overhead with the latest generic RH mitigation mechanisms in the literature in Table 2. For this experiment, our strategy is to consider the same 32GB: 16-bank of DDR4 DRAM for all frameworks and normalize the capacity overhead and area overhead across different frameworks. In Table 2, *i)* the involved memory indicates the type of memory used by the framework for RH protection. As discussed, certain frameworks rely on a counter to monitor intrusions and store tracking information in the system using CAM/SRAM. Nonetheless, because of the considerably higher cost of CAM and SRAM in comparison to DRAM, selecting a framework with such supplementary resources may be controversial. For example, Table 2 shows that only Graphene [11] and TWiCE [8] occupy two fast storage resources simultaneously. At the same time, Hydra [12], SRS [21], and RRS [16] rely on SRAM in addition to DRAM. *ii)* Capacity overhead refers to the memory

Table 2: Comparison with prior generic RowHammar mitigation frameworks.

Framework	involved memory	capacity overhead	area overhead
Graphene [11]	CAM-SRAM	$0.53MB^{\ddagger}+1.12MB^{\dagger}$	1 counter
Hydra [12]	SRAM-DRAM	$56KB^{\dagger}+4MB^{*}$	1 counter
TWiCE [8]	SRAM-CAM	$3.16MB^{\dagger}+1.6MB^{\ddagger}$	1 counter
Counter per Row	DRAM	$32MB^{*}$	16384 counters
Counter Tree [19]	DRAM	$2MB^{*}$	1024 counters
RRS [16]	DRAM-SRAM	$4MB^{*}+NR^{\dagger}$	NULL
SRS [21]	DRAM-SRAM	$1.26MB^{*}+NR^{\dagger}$	NULL
SHADOW [20]	DRAM	$0.16MB^{*}$	0.6%
P-PIM [25]	DRAM	$4.125MB^{*}$	0.34%
DNN-Defender	*DRAM*	***0***	***0.02%***

NR = Not Reported

*The capacity overhead of DRAM. †The capacity overhead of SRAM. ‡The capacity overhead of CAM.

resources utilized by RH in a framework. Such resources are dedicated solely to RH and not for other purposes. Take Graphene [11] as an example, it requires storing counting tables in SRAM (1.12 MB), and the space occupied by these tables can no longer be used as a shared storage space to store data. Graphene also requires 0.53 MB CAM to track vulnerable rows. Take SHADOW [20] as an example, 0.16 MB capacity of DRAM is dedicated to enabling RH mitigation. Our framework stands out from others as it does not utilize any additional memory resources for RH mitigation. Unlike Graphene [11], TWiCE [8], SRS [21], and RRS [16], DNN-Defender does not require any fast-read memory. Furthermore, in contrast to SHADOW [20] and P-PIM [25], DNN-Defender does not even sacrifice DRAM resources, where all rows can be used for storing data in the same way as ordinary rows. *iii)* Some frameworks not only require storage device resources but also additional components for RH mitigation. We can find that DNN-Defender offers one of the most area-efficient solutions compared to other frameworks.

Security & Performance Analysis. We establish that the system's security level is directly proportional to the time taken by an attacker to breach it, i.e., the longer it takes to breach the system, the higher the security level. The first assumption is that the data has no unique mapping, so we consider the vulnerable data rows to be evenly distributed in all banks. Therefore, the number of data rows we need to protect in each bank is given by $N_s = S_{bit}/banks$, where N_s is finite. In other words, when the number of under-attack rows increases, N_s will exceed the defendable threshold. It is, therefore, essential to identify the threshold that our framework can withstand. Assuming the worst-case scenario in which a row contains only a single target weight bit, the maximum number of defended BFA corresponds to the number of target rows. As DNN-Defender doesn't modify the circuitry of the original DRAM array, we can utilize the timing baseline of DRAM without any alterations as given in [20] to calculate the time required for swap operation, where $T_{swap} = 3 \times T_{AAP}$, T_{AAP} = 90ns. As discussed, T_{RH} depends on the DRAM process node as the minimum number of activations that can impose bit-flip. So the time constraint for DNN-Defender to perform the swap operations is given by $T_{ACT} \times T_{RH}$, and the maximum number of swap operations can be calculated as $\frac{T_{ACT} \times T_{RH}}{T_{swap}}$. We can calculate $T_n = T_{ACT} \times T_{RH} + T_{swap} \times N_s$, and then find the total number of swap operations in single T_{ref} by $N = \frac{T_{ref}}{T_n} \times N_s$. Our experiments show that DNN-Defender and SHADOW [20] are the *only frameworks that can withstand the white-box attacks*, therefore, we only report their security level analysis. Our results indicate that even the SRS mechanism [21] cannot defend against white-box attacks for a period of one day. Figure 8(a) reports the time-to-break in days for various T_{RH}s and the corresponding number of the defended BFAs. We observe that our framework outperforms SHADOW in 1k, 2k, 4k, and 8k thresholds, e.g., as indicated for T_{RH}=4k, the attacker will require ~1180 days to break a DNN-Defender-supported system, while SHADOW stands up for ~894 days.

Figure 8(b) shows DNN-Defender and SHADOW [20] with 1k, 2k, 4k, and 8k RH thresholds. As mentioned earlier, we pick SHADOW as one of the best RH mitigation mechanisms over others for this comparison. The figure illustrates that with an increase in the number of BFAs, the rate of latency increase decelerates and eventually reaches a limit for both frameworks. We chose four critical points of BFAs, i.e., 7K, 14K, 28K, and 55K, where each corresponds to the maximum allowable number of BFAs under various thresholds. This selection is made to facilitate a comparative analysis. Considering a 4k threshold, an escalation in the number of BFAs leads to a peak in latency. When compared to SHADOW operating at the same threshold, our framework exhibits lower latency in all cases. We consider the power consumption of a standard DRAM process as a benchmark to fully reveal all aspects of our proposed framework. Compared with other frameworks, DNN-Defender indicates no significant power-saving. For instance, even when SHADOW is set with a threshold of 1k, DNN-Defender shows a negligible 1.6% power-saving. However, considering the power consumption by SRAM-based frameworks such as SRS and RRS and off-chip memory communication in such systems, DNN-Defender offers a significant improvement (3.4× compared with SRS).

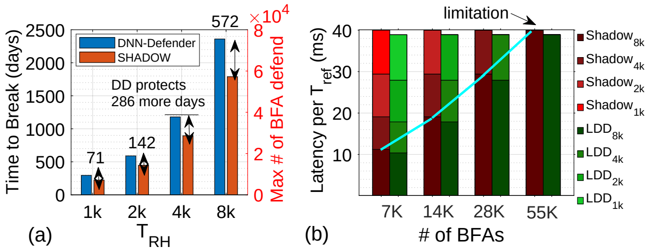

Figure 8: (a) Time-to-break DNN-Defender (DD) and SHADOW in different RowHammer thresholds, (b) Latency of DNN-Defender and SHADOW [20] at different no. of BFA.

5.2 Defense Evaluation

Evaluation of DNN-Defender against Semi-White-Box BFA. First, we consider a naive BFA attack [13] where the attacker is not aware of our defense strategy (Semi-White-box). Such a naive attack will fail since the targeted bit-flip sequence is not optimized to bypass our defense. A naive attacker will generate a small sequence of target bits and will attempt to flip them. However, our defense will eliminate the impact of bit-flips via the swap operation, and the attacker will not achieve any success (i.e., accuracy degradation) using the existing BFA algorithm.

Evaluation of DNN-Defender against White-Box BFA. With complete white-box knowledge, the attacker is aware of our defense and tries to evade it through adaptive search. To bypass DNN-Defender, the attacker can generate multiple sequences of targeted bits offline using a copy of the target model and evaluate the attack's success by launching the attack to the victim space. However, if a specific chain of bit sequences fails to degrade model accuracy, the attacker can skip this sequence to generate a new set of bit sequences for the next attack round. This way, the attacker will iterate through all the Secured Bits (SB) protected by DNN-Defender and still observe no success i.e., accuracy drop at the output.

Figure 9 demonstrates the effectiveness of DNN-Defender in mitigating the performance degradation caused by BFA when the attacker further adapts the search and attempts to flip more bits than protected by our defense (denoted as SB + # of additional bit-flips). The plots show multiple curves of performance degradation on each of the evaluation models (e.g., VGG-11, ResNet-18, ResNet-34), with each curve representing the degradation after securing a specific

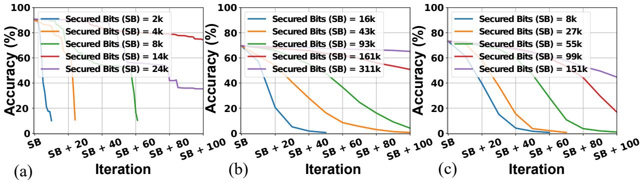

Figure 9: DNN-Defender evaluation for different amounts of Secured Bits (SB) for (a) VGG-11 trained on CIFAR-10, (b) ResNet-18 trained on Imagenet, and (c) ResNet-34 trained on Imagenet against BFA.

number of bits using DNN-Defender denoted as Secured Bits (SB). The plots reveal that as the SB increases, it takes the attacker an increasing number of iterations to cause the same performance degradation, gradually deteriorating the performance of BFA near the random attack level. For example, in Fig. 9, for the VGG-11 model, increasing the secured bits from 2k to 8k increases the number of additional bit-flips required to achieve the same attack efficacy by ∼ 6×. Eventually, even this adaptive white-box BFA attack loses its potency after securing 24k bits (still only ∼ 4% of the total model bits). In conclusion, our proposed DNN-Defender will increase the attack time and effort by requiring an increasing amount of bit-flips to achieve the same attack efficacy as the baseline (no defense). If we secure a large number of bits (e.g., ∼ 24k for VGG-11), then even after increasing the bit-flip, the effect of BFA can be reduced to a random attack level.

Comparison to BFA Defenses. In Table 3, we compare DNN-Defender against the existing *training-based DNN software defenses* [3, 9, 14] and *selected generic hardware defenses*, i.e., RRS [16], SRS [21], and SHADOW [20]. A general approach among the prior software-based defense works is to reduce model weight bit-width precision [3, 4] and increase the model size to reduce the impact of weight noise on accuracy [14]. Here, a binary neural network [14] with both binary weight and activation achieves the best defense performance against the BFA, requiring over 1000 bit-flip to reduce the model accuracy close to random guesses. When protecting the exact required number of (e.g., 1150) vulnerable bits, DNN-Defender can resist the BFA attack better than the binary model. However, our attack incurs slight hardware (e.g., latency & energy) overhead which is not the case for software training algorithms [3, 9, 14]. In contrast, all the software-based training methods suffer from high training overhead and model accuracy drop. Our method is an effective defense against BFA w/o requiring any training overhead or performance drop with minimal hardware overhead. Additionally, any software protection [10] or DNN training algorithm [3, 14] is not necessarily a competing method against our defense; prior training-based defenses can further boost the protection against BFA on top of our method. In addition, we observe that DNN-Defender shows higher post-attack accuracy compared with prior designs withstanding more number of BFAs.

Table 3: Comparison to BFA software defences on CIFAR-10 evaluated attacking a ResNet-20 model.

Models	Clean Acc.(%)	Post-Attack acc.(%)	Bit-Flips #
Baseline ResNet-20 [13]	91.71	10.90	20
Piece-wise Clustering [3]	90.02	10.09	42
Binary weight [3]	89.01	10.99	89
Model Capacity × 16 [14]	93.7	10.00	49
Weight Reconstruction [9]	88.79	10.00	79
RA-BNN [14]	90.18	10.00	1150
RRS [16]	91.71	75.65	342
SRS [21]	91.71	75.92	378
SHADOW [20]	91.71	88.80	985
DNN-Defender	***91.71***	***91.71***	***1150***

6 CONCLUSIONS

Herein, we proposed a powerful DRAM-based defense mechanism called DNN-Defender that leverages the potential of in-DRAM swapping to protect quantized DNNs from targeted RowHammer bit-flip attacks. Our results indicate that DNN-Defender is capable of providing robust protection against consecutive targeted RowHammer attacks on CIFAR-10 and ImageNet datasets downgrading its performance to a random attack level.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation (NSF) under grant no. 2228028.

REFERENCES

[1] 2011. *NCSU EDA FreePDK45.* http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[2] Nathan Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39 (2011), 1–7.

[3] Zhezhi He et al. 2020. Defending and harnessing the bit-flip based adversarial weight attack. In *CVPR*. 14095–14103.

[4] Sanghyun Hong et al. 2019. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks.. In *USENIX Security Symposium*. 497–514.

[5] Patrick Jattke et al. 2022. Blacksmith: Scalable rowhammering in the frequency domain. In *SP*. IEEE, 716–734.

[6] Jeremie S Kim et al. 2020. Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques. In *ISCA*. IEEE, 638–651.

[7] Yoongu Kim et al. 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.

[8] Eojin Lee et al. 2019. TWiCe: Preventing row-hammering by exploiting time window counters. In *ISCA*. 385–396.

[9] Jingtao Li et al. 2020. Defending bit-flip attack through DNN weight reconstruction. In *DAC*. IEEE, 1–6.

[10] Jingtao Li et al. 2021. RADAR: Run-time Adversarial Weight Attack Detection and Accuracy Recovery. *arXiv preprint arXiv:2101.08254* (2021).

[11] Yeonhong Park et al. 2020. Graphene: Strong yet lightweight row hammer protection. In *MICRO*. IEEE, 1–13.

[12] Moinuddin Qureshi et al. 2022. Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking. In *ISCA*.

[13] Adnan Siraj Rakin et al. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *ICCV*. 1211–1220.

[14] Adnan Siraj Rakin et al. 2021. Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy. *arXiv preprint arXiv:2103.13813* (2021).

[15] Adnan Siraj Rakin et al. 2021. T-bfa: Targeted bit-flip adversarial weight attack. *IEEE TPAMI* 44, 11 (2021), 7928–7939.

[16] Gururaj Saileshwar et al. 2022. Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows. In *ASPLOS*. 1056–1069.

[17] Vivek Seshadri et al. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *MICRO*. IEEE, 273–287.

[18] Vivek Seshadri, Yoongu Kim, et al. 2013. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In *MICRO*.

[19] Seyed Mohammad Seyedzadeh et al. 2016. Counter-based tree structure for row hammering mitigation in DRAM. *CAL* 16 (2016).

[20] Minbok Wi et al. 2023. SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In *HPCA*. IEEE, 333–346.

[21] Jeonghyun Woo et al. 2022. Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. *arXiv preprint arXiv:2212.12613* (2022).

[22] Mengjia Yan et al. 2020. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *USENIX Security Symposium*.

[23] Fan Yao et al. 2020. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *USENIX*.

[24] Ranyang Zhou et al. 2022. LT-PIM: An LUT-Based Processing-in-DRAM Architecture With RowHammer Self-Tracking. *IEEE Computer Architecture Letters* 21, 2 (2022), 141–144.

[25] Ranyang Zhou et al. 2023. P-PIM: A Parallel Processing-in-DRAM Framework Enabling Row Hammer Protection. In *DATE*. IEEE.