

DRAM-Locker: A General-Purpose DRAM Protection Mechanism against Adversarial DNN Weight Attacks

Ranyang Zhou[†], Sabbir Ahmed[‡], Arman Roohi^{*}, 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

^{*}School of Computing, University of Nebraska–Lincoln, Lincoln NE, USA

rz26@njit.edu, sahmed9@binghamton.edu, aroohi@unl.edu, arakin@binghamton.edu, shaahin.angizi@njit.edu

Abstract—In this work, we propose DRAM-Locker as a robust general-purpose defense mechanism that can protect DRAM against various adversarial Deep Neural Network (DNN) weight attacks affecting data or page tables. DRAM-Locker harnesses the capabilities of in-DRAM swapping combined with a lock-table to prevent attackers from singling out specific DRAM rows to safeguard DNN’s weight parameters. Our results indicate that DRAM-Locker can deliver a high level of protection downgrading the performance of targeted weight attacks to a random attack level. Furthermore, the proposed defense mechanism demonstrates no reduction in accuracy when applied to CIFAR-10 and CIFAR-100. Importantly, DRAM-Locker does not necessitate any software retraining or result in extra hardware burden.

I. INTRODUCTION

The widespread progress of Deep Neural Networks (DNN), achieving unparalleled performance and high accuracy even with models that have low bit-widths, has recently led to the emergence of various security-related attacks across many deep learning applications [1], [2]. Recent research indicates that by identifying and manipulating a small set of susceptible bits within well-trained DNN weight parameters, adversaries can significantly degrade the resulting accuracy of these models as indicated in Fig. 1(a) [2]. These attacks, known as Bit-Flip Attacks (BFAs), have primarily been facilitated by a phenomenon in DRAM called RowHammer as the manifestation of a DRAM cell-to-cell interference and failure mechanism [3], [4]. On top of this, a novel class of adversarial fault injection techniques has been recently introduced [5]–[7] that exploits BFA in memory addresses. Here an attacker can leverage the RowHammer attack to flip the bits in the page tables to corrupt the translation between the virtual and physical memory addresses. This gives attackers system-level privileges to overwrite a specific data block stored in a physical address using a replacement data block stored at a different physical memory address [5].

The RowHammer attack occurs when a malicious process repetitively activates and pre-charges a specific row (referred to as the aggressor row) until it reaches a certain threshold (T_{RH}) [7]–[9]. This repeated activation induces bit-flips in adjacent rows (referred to as victim rows). Unfortunately, due to the shrinking size of DRAM chips in modern manufacturing processes, DRAM has become increasingly susceptible to RowHammer-induced bit-flips [10]. The data reported in Fig. 1(b) shows a notable downward trajectory in the T_{RH} over recent years. For instance, compared to DDR3 (new), LPDDR4 (new) requires approximately 4.5 times fewer hammering iterations to trigger the same effect [11]. This trend suggests that T_{RH} will nearly vanish with the advent of DDR5 [12]. To prevent RowHammer attacks, DRAM manufacturers

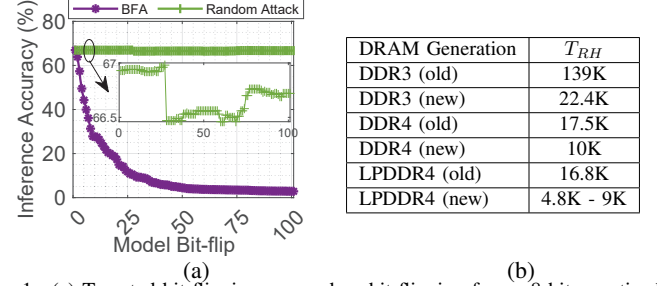


Fig. 1. (a) Targeted bit flipping vs. random bit flipping for an 8-bit quantized VGG11 trained on CIFAR100, (b) RowHammer thresholds [11].

and researchers have proposed hardware-based victim-focused defense mechanisms to proactively refresh the victim rows [3], [7]. However, such RowHammer mitigation proposals have faced a huge overhead both from latency and power consumption perspectives [13]. To mitigate this, recent aggressor-focused swap-based mechanisms [11], [14] proactively swap and unswap aggressors with random rows before reaching the T_{RH} . This raises another issue, i.e., the challenge of precisely monitoring the targeted row. SHADOW [15] leverages unintelligent swap operations on all potential target rows to safeguard them from Rowhammer attacks. However, it is evident that attackers do not typically target numerous rows simultaneously, resulting in the wasteful execution of swap operations.

While there has been a multitude of generic victim-focused [3], [16] and aggressor-focused [11], [14], [17] mechanisms to protect the memory rows against RowHammer attacks, a few works have tried to tailor defense mechanisms for adversarial DNN weight attacks in software. Most of these methods try to make the model more robust by either using fewer bits for the model’s weights [18] or by making the model larger to reduce the effect of weight noise on accuracy. Such mechanisms require software retraining or result in extra hardware burdens. In this work, we propose DRAM-Locker as a novel and advanced DRAM defense mechanism to protect the DNN against both adversarial BFA and page table attacks targeting weight parameters. Our contribution is summarized as follows.

- We are the first to demonstrate a general-purpose DRAM defense mechanism with a lock-table that protects memory against DNN weight attacks affecting data or page tables;
- We develop ISA, software support, and the interface required to implement in-DRAM swap operation; and
- We extensively analyze the DRAM-Locker’s applicability and efficiency in withstanding RowHammer vulnerability compared to recent hardware/software techniques over CIFAR-10 and CIFAR-100 DNN datasets.

II. BACKGROUND & MOTIVATION

A. 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” [19]. 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. Taking advantage of the DRAM’s ability to transfer a complete data row to the corresponding row buffer during read operation, RowClone [20] has been devised as an uncomplicated and highly effective technique for facilitating a rapid in-memory copy operation (completed in less than 100 nanoseconds) within the DRAM sub-array, allowing data to be transferred from a source row to a destination row. 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 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 [20].

B. SOTA Defense Mechanisms

Multiple software and hardware mitigation mechanisms have been proposed to reduce the impact of RowHammer-based attacks. The standard mitigation approach used by manufacturers such as Apple [21] is to reduce the refresh period, e.g., from 64ms to 32ms. The system manufacturers tend to increase refresh rates and hardware RHP [10]. Along this line, Target Row Refresh (TRR) [7] and other counter-based detection methods [17], [22] require add-on hardware to calculate rows’ activation and record it to other fast-read-memory (SRAM [23]/CAM [24]). The controller will then refresh the target row if the number reaches T_{RH} [7]. However, such proactive

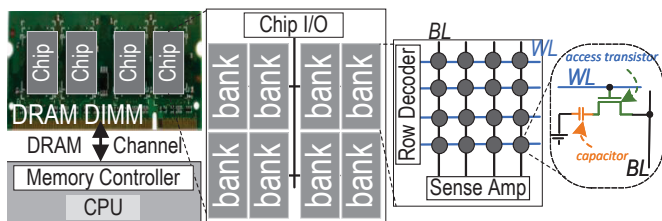


Fig. 2. Organization of a DRAM chip.

refreshing proposals have faced a huge overhead both from latency and power consumption perspectives. A method called Secure Row-Swap (SRS) [11] has demonstrated the use of fewer counters for crucial data and implemented associated threat mitigation using the swap operation. However, such mitigations not only impose a significant slowdown to the system but also require a pre-defined threshold at CPU design time. To solve this issue, LT-PIM protects only a critical part of the memory rather than the complete data [13]. Mitigations of this kind are susceptible to breakthrough attacks on many bits, such as Half-Double [25], which take advantage of previously unknown access patterns. Software-only schemes offer the advantage of compatibility with existing hardware, making them more deployable. However, it's important to note that existing software-only mitigations require modifications to the memory allocator and may not be fully effective. The software-based efforts such as SoftTRR [26] enable software tracking of activations to Page Table Entry (PTE) rows and accordingly issue mitigation. Similar to hardware-based TRR, the efficacy of such defense reduces when the attack happens on more number of bits and larger distances as in Half-Double. CTA [27] is susceptible to privilege escalation attacks such as PThammer [6] targeting L1PTE. There are a few defense mechanisms developed especially for exploits on page tables. SecWalk [28] can detect only up to 4-bit flips per PTE with error detection codes though it is susceptible to attacks like ECCploit [29]. PT-Guard [5] presents a method that tracks the rows under attack by asserting the Message Authentication Code (MAC) to the PTE. In DRAM writes, they split the MAC into eight parts and embed them in the unused bits of the PTE. Then they recomputed the MAC and performed integrity checks on the hardware page table to ensure the PTE remained unchanged. In our work, there is no need to occupy the resources of software sides without reducing the performance.

III. THREAT MODEL

The DRAM-Locker is designed to safeguard DRAM against the following adversarial DNN weight attacks.

Bit-Flip Attack (BFA). We assume the following threat model for BFA. 1) DNN model inference is running on a resource-sharing environment which is practical due to the recent popularity of Machine-Learning-as-a-Service (MLaaS) [30]. The attacker can run user-level un-privileged processes remotely on the same machine deployed by the victim DNN model; 2) Each DRAM 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 as depicted in Fig. 3(a); 3) 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; 4) The attacker has a detailed mapping file 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) [15]; and 5) the attacker is aware of the internal structure of the DNN models, e.g., the number of layers and the width

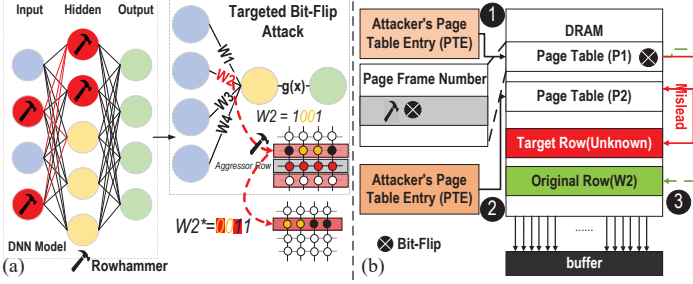


Fig. 3. Two major DNN threat models in this work: (a) BFA, (b) PTA.

of each layer. On top of that, the attacker has complete knowledge of the DNN model parameters, their values, and bit representation for inference. They can catch the bits that can mostly reduce the accuracy, such as $W2$ shown in Fig. 3(a).

Page Table Attack (PTA). We assume the following practical threat model for PTA established by prior works [5]. 1) The attacker can map the virtual addresses to physical addresses using several techniques such as leveraging huge page support, hardware-based side-channel attack [31], and memory messaging [32]; 2) The attacker can cause a targeted bit-flip to the page table and cause a bit-flip at the desired location using fast and precise multi-bit-flip techniques [33]; 3) We assume the kernel and operating system are trusted and well-protected; 4) Following standard practice, we assume the commercial DRAM is not protected by ECC and cannot protect large-scale deep learning models against RowHammer [33]; and 5) Attacker can insert their own virtual page table entry, as depicted in Fig. 3(b) ① and ② perform a bit-flip within their page table entries P1, coercing them to direct to a secondary virtual address row P2 [5]. Knowing the location of the target row after flipping the page table, the attacker can redirect the pointer to the target row out of range.

IV. DRAM-LOCKER

The DRAM-Locker aims to serve as a comprehensive remedy for various forms of severe RowHammer attacks targeting DNN weight parameters. The core idea behind DRAM-Locker is to prevent RowHammer attackers from singling out specific DRAM rows by securing those rows with locks. DRAM-Locker employs the SWAP command to interchange the locked row with a free one, effectively unlocking it and restoring normal program execution. Compared with the previous counter-based designs [7], [22], DRAM-Locker only requires a fast read-and-write memory space for the lock-table without any counter overhead. The DRAM-Locker also offers scalability and flexibility w.r.t. various applications. In other words, our framework allows users to customize the data they are willing to protect without requiring changes to the framework. As compared with other swap-based frameworks such as [15], DRAM-Locker is more directional and can effectively reduce the overhead of useless and time-consuming protection.

A. Protection Framework

Our protection framework exploits the lock-table to record the physical addresses of the DRAM that need to be locked. These addresses depend on the physical address of the data the user is willing to protect. For example, frequently used data is often the target of attackers, so adjacent rows of these

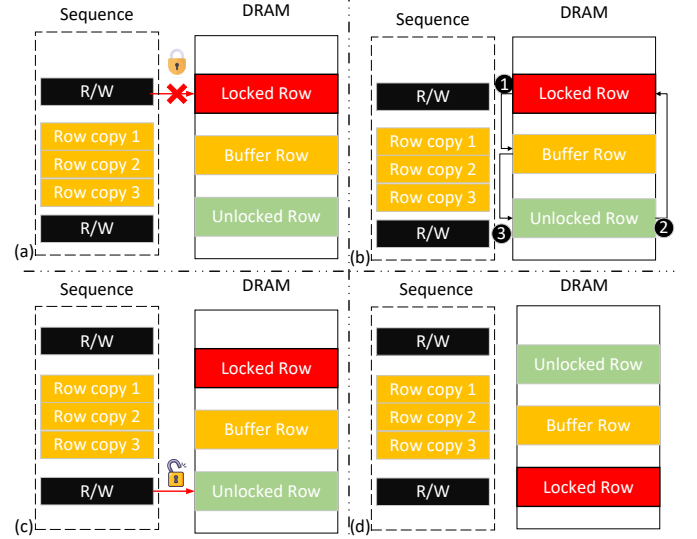


Fig. 4. Overview of the DRAM-Locker. (a) R/W of the Locked Row, (b) Implementation of a SWAP operation, (c) R/W of the Unlocked Row, (d) Final status after updating the lock-table.

data can be recorded in a lock-table. In addition, due to the uncertainty of attacks, users can manually add any row that has a high probability of becoming an aggressor row into the the lock-table. As depicted in Fig. 3(a), conventional BFA techniques have been proven to be highly inefficient when applied to large-scale DNNs with extensive datasets. They often perform a large number of bit-flips on weights and/or activations without effectively reducing the accuracy of the model. But if the attacker focuses on attacking adjacent rows of particular weights to cause bit-flip, this will cause fatal damage to the DNN model. We propose to segregate and retain these addresses within a lock-table in SRAM. Once the addresses are stored in the lock-table, any attempt to access this data without the accompanying unlock command will result in a block. In DNNs, weights typically represent high-frequently used data. Locking these weights can lead to substantial delays and a significant burden on the controller. Additionally, frequent access implies continuous refreshing, and the possibility of bit-flips occurrence within these rows is definitely low. If we choose to lock frequently used rows, then we have to unlock them constantly, so locking adjacent rows can substantially decrease the need for frequent unlocks.

DRAM-Locker prepares for subsequent lock recognition and unlocking operations by storing instructions in Sequence. As shown in Fig. 4(a), if an R/W (Read/Write) instruction in the Sequence contains the locked address, access to the Locked Row will be denied. Therefore, the DRAM-Locker needs SWAP operations to unlock. As shown in Fig. 4(b), SWAP operations are transferred into DRAM instructions via ISA. The DRAM implements the sets of Row Copy instructions in order to perform the SWAP operation to pull out the Locked Row. Besides, since the RowClone principle is to overwrite the data of the copied row into a new row, we devise a Buffer Row. The detailed steps are as follows. In step ①, DRAM-Locker copies the Locked Row to the Buffer Row; in step ②, it copies the Unlocked Row to the Locked Row; and in

step ③, DRAM-Locker copies the Buffer Row to the Unlocked Row. This step does not change the lock-table, but exchanges the original data of the two addresses, ultimately achieving the unlocking process. The data stored at the locked address has been swapped into an unlocked row. Fig. 4(c) shows that the next R/W instruction requires the address of the Unlocked Row to access the data of the original Locked Row. Taking the T_{RH} into account, it becomes necessary to re-secure the previously unlocked rows by periodically refreshing the lock-table. In Fig. 4(d), for instance, following the completion of a SWAP operation, the controller reinstates the swapped address into the lock-table after a cumulative count of 1k R/W instructions, effectively re-locking the data row.

B. Lock-table & Sequencing

The lock-table plays a crucial role in our platform similar to that of the count-table in state-of-the-art designs [22], [23]. Meanwhile, lock-table has no dependency on extra components such as counters. While counter-based designs employ a count table to monitor memory access patterns and keep track of the number of accesses to each row in the memory, DRAM-Locker stores the addresses of vulnerable rows and denies permissionless access requests to them. In counter-based designs, when a row is accessed excessively within a short time frame, the count-table is the indicator of a potential RowHammer attack. In response, the defense can take preventive actions to mitigate it. In DRAM-Locker, whenever the controller recognizes an R/W instruction, it will traverse the lock-table. If the address contained in the instruction is found, the instruction will be skipped. Therefore, no matter how many requests the attacker sends, they will be invalid and the instructions will not be executed. As a result, the latency caused by invalid instructions will also be eliminated. The instructions from attackers will also be stored in Sequence. If these instructions are skipped, we can also improve the performance. When the program necessitates accessing these data rows, the SWAP operation is required to unlock the secured rows. So we insert row copy instructions to the sequence in order to execute the SWAP operations. To realize this, DRAM-Locker requires ISA support which will be introduced in the next subsection. DRAM-Locker uses three Row Copy instructions to implement the SWAP operation.

C. ISA Support

The DRAM-Locker is intentionally crafted as a general-purpose autonomous defense mechanism. Consequently, it needs to be accessible to system-level libraries and developers. From a developer's perspective, DRAM-Locker needs to transfer the SWAP operations and insert some specific instructions to perform the defense. Therefore, to enable widespread RowHammer defense for both BFA and PTA, ISA support is indispensable for any user-level program. This can be translated into the DRAM-Locker's hardware instruction set during installation. As shown in Fig. 5, DRAM-Locker is designed to process two 16-bit instruction types after compiling the upper-level code: (1) a copy instruction based on RowClone method [20], (2) an instruction for control operations. When OP is 01, DRAM-Locker performs a row copy operation by activating both μ Reg src. and μ Reg des. Opcodes 10 and 11

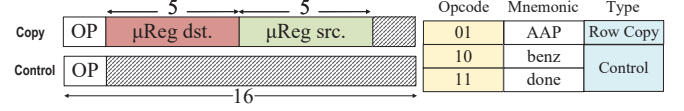


Fig. 5. DRAM-Locker's instructions, μ Ops, and their description.

correspondingly denote straightforward control operations for managing loops and termination within the DRAM-Locker's control flow., i.e., benz, done.

D. Discussion: Challenges with Unsuccessful Swapping.

We conducted comprehensive circuit-level simulations to investigate the impact of process variations of in-DRAM SWAP as the key operation in the DRAM-Locker based on the framework discussed in Section V. Our study considered a worst-case DRAM cell scenario where all components (including cell/BL/WL capacitance and transistor in Fig. 2) exhibited variation. We performed a Monte Carlo simulation using the 45nm NCSU PDK library. This simulation comprised 10,000 trials, and we systematically increased the variation in parameters from $\pm 0\%$ to $\pm 20\%$. It is worth noting that reducing the transistor size is anticipated to exacerbate the impact of process variation. Our observation shows that the percentage of erroneous SWAP operation across 10,000 trials is 0%, 0.14%, and 9.6% for $\pm 0\%$, $\pm 10\%$, and $\pm 20\%$, respectively.

V. PERFORMANCE EVALUATION

Setup & Framework. We present a cross-layer evaluation framework as depicted in Fig. 6 to demonstrate the benefits of DRAM-Locker in protecting memory against adversarial DNN weight attacks. Firstly, we develop DRAM-Locker's sub-arrays with peripherals using Cadence Spectre in the 45nm NCSU PDK library [34] at the circuit level to verify SWAP functionality, attain performance parameters, and measure the row-shuffle time. The memory controller and registers are 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. Next, we implemented DRAM-Locker's ISA in gem5 [35], and exported the memory statistics and performance to an in-house C++ DRAM-Locker 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 and PTA using various DNN models and datasets, where the weights are quantized to 8-bit width. To carry out the BFA and PTA, we randomly sampled images from the test/validation set, with a default sample size of 128 for both datasets.

Hardware Overhead Analysis. We compare the DRAM-Locker's hardware overhead with the latest RowHammer mitigation mechanisms in the literature in Table I. In this experiment, we utilize a uniform configuration of 32GB:16-bank DDR4 DRAM for all frameworks. The aim is to standardize capacity and area overheads across different frameworks. In Table I, i) the involved memory refers to the type of memory utilized by the framework for RowHammer protection. As previously discussed, certain frameworks rely on counters to monitor intrusions and store tracking information within

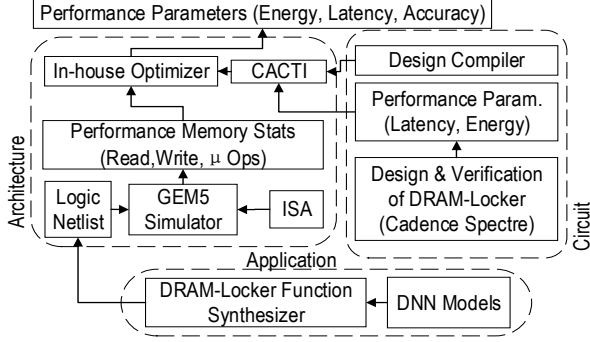


Fig. 6. Proposed cross-layer evaluation framework.

the system, utilizing Content-Addressable Memory (CAM) or Static Random-Access Memory (SRAM). However, CAM and SRAM are significantly more costly in comparison to DRAM with the same overhead. Therefore, opting for a framework with such additional resources may lead to debates regarding cost-effectiveness. Take the Graphene [24] as an example, it employs both CAM and SRAM and the total capacity overhead is 0.53MB + 1.12 MB. It is worth pointing out that while most of the frameworks' area overhead in Table I is already quite compact, in terms of practical circuit design, minimizing alterations to the existing structure is generally more pragmatic. We observe that SHADOW [15] and DRAM-Locker promote the use of extremely less extra components for constructing their defenses. Therefore, these two frameworks are selected for further security and performance analysis in the next part.

Security & Performance Analysis. Figure 7(a) shows the comparison between latencies of SHADOW [15] with $T_{RH}=1k$, 2k, 4k, and 8k, and the DRAM-Locker with $T_{RH}=1k$. The threshold here represents the number of necessary visits to the aggressor row by the attacker to ensure that the victim row can generate a bit-flip. Whether in PTA or BFA, the attacker must ensure that the bit they expect is flipped accurately, so in the experiment, we consider the threshold as the minimum number of times. Also in this experiment, we assume that DRAM-Locker suffers from a 10% error rate due to unsuccessful SWAPs of instructions. Please note that if the DRAM-Locker shows no errors then it will be ideally invulnerable. In Fig. 7(a), due to the DRAM-Locker's distinctive SWAP mechanism, we exclusively evaluate the worst case, which is the 1k threshold. In this case, our framework demands the highest number of SWAP operations, resulting in the longest latency. We can see that SHADOW [15] exhibits a defense threshold. System integrity is compromised once this threshold is surpassed, signifying that further delay escalation is halted. Conversely,

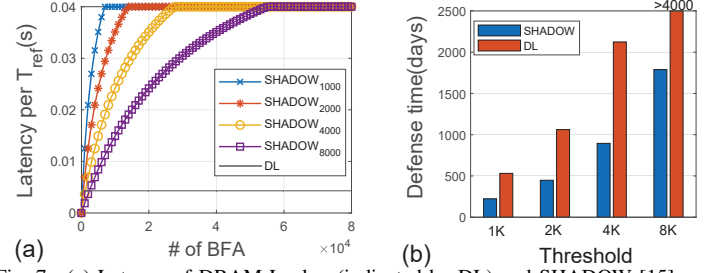


Fig. 7. (a) Latency of DRAM-Locker (indicated by DL) and SHADOW [15] in different numbers of BFA, (b) Defense time (per day) in various thresholds.

DRAM-Locker operates without such a defense threshold. Hence, the latency in DRAM-Locker is solely contingent on T_{RH} , with the worst case in our model being a $T_{RH}=1k$. Unlike SHADOW, our framework not only boasts remarkably low latency but also demonstrates general applicability across various DRAM chips, without any distinctions. In Fig. 7(b), we conducted experiments to evaluate the duration for which SHADOW and DRAM-Locker can effectively defend against attacks. Considering a 10% error rate during the execution of Row Copy instructions, we take the expected outcomes of these error instructions into account. Then, we deemed success as achieving a probability exceeding 99%. This means in both PTA and BFA threat models, when the attacker aims to flip the target bit, the probability is lower than 1%. As a result, we observe that even considering a 10% error to each row copy, DRAM-Locker could maintain an effective defense mechanism against attacks for a period exceeding 500 days under the 1K threshold. Therefore, in contrast to SHADOW, our framework exhibits reduced latency and extended defense duration.

Evaluation of DRAM-Locker against BFA & PTA. Figure 8 shows the efficacy of DRAM-Locker in alleviating the performance degradation caused by BFA. Note that this evaluation considers the worst case $\pm 20\%$ variation on all DRAM components and thus considers that the BFA is successful 9.6% of the time as discussed before. The figure presents performance degradation across two different evaluation models, ResNet-20 trained on the CIFAR-10 dataset and VGG-11 trained on the CIFAR-100 dataset. The plots clearly reveal that with DRAM-Locker, it takes the attacker an increasing number of iterations to cause the same performance degradation. Thus, DRAM-Locker significantly increases the computational overhead for the attacker by necessitating more bit-flips to achieve equivalent attack effectiveness. We ran the same experiment considering PTA as the attack mechanism. The findings reveal that the attacker similarly needs a growing number of iterations to induce an equivalent performance decline.

Comparison to other Defenses. In Table II, we compare our proposed DRAM-Locker, with existing training-based defenses [18], [37], [38]. Most of these methods try to make the model more robust by either using fewer bits for the model's weights [18] or by making the model bigger to reduce the effect of weight noise on accuracy [37]. When we use DRAM-Locker to protect a specific number of vulnerable bits (for example, 1150), it does a better job of defending the BFA than the binary model. However, it's worth noting that our method does add a small amount of delay and energy use,

TABLE I
COMPARISON WITH PRIOR ROWHAMMAR MITIGATION FRAMEWORKS.

Framework	involved memory	capacity overhead	area overhead
Graphene [24]	CAM-SRAM	0.53MB [†] +1.12MB [†]	1 counter
Hydra [17]	SRAM-DRAM	56KB [†] +4MB*	1 counter
Twice [23]	SRAM-CAM	3.16MB [†] +1.6MB [‡]	1 counter
Counter per Row	DRAM	32MB*	16384 counters
Counter Tree [22]	DRAM	2MB*	1024 counters
RRS [14]	DRAM-SRAM	4MB*+NR [†]	NULL
SRS [11]	DRAM-SRAM	1.26MB*+NR [†]	NULL
SHADOW [15]	DRAM	0.16MB*	0.6%
P-PIM [36]	DRAM	4.125MB*	0.34%
DRAM-Locker	DRAM-SRAM	0+56KB [†]	0.02%

NR = Not Reported

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

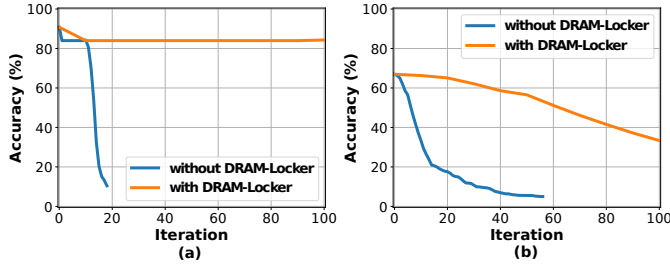


Fig. 8. DRAM-Locker evaluation for (a) ResNet-20 trained on CIFAR-10, (b) VGG-11 trained on CIFAR-100. The degradation in performance is shown for 100 iterations of the attack.

which isn't an issue for the training-based defenses [18], [37], [38]. On the other hand, the existing training-based methods often come with downsides like taking a lot of time to train and reducing the model's accuracy. Our DRAM-Locker doesn't have these problems; it works well against BFAs and PTAs without needing extra training time or hurting performance, and only adds a minor hardware cost. Plus, our method can be used alongside existing software protections or training-based defenses [18], [37] to make the model even more secure against various types of attacks.

TABLE II

COMPARISON TO OTHER COMPETING SOFTWARE DEFENSE METHODS ON CIFAR-10 DATASET EVALUATED ATTACKING A RESNET-20 MODEL.

Models	Clean Acc.(%)	Post-Attack acc.(%)	Bit-Flips #
Baseline ResNet-20 [2]	91.71	10.90	20
Piece-wise Clustering [18]	90.02	10.09	42
Binary weight [18]	89.01	10.99	89
Model Capacity \times 16 [37]	93.7	10.00	49
Weight Reconstruction [38]	88.79	10.00	79
RA-BNN [37]	90.18	10.00	1150
DRAM-Locker	91.71	91.71	1150

VI. CONCLUSIONS

Here, we proposed a general-purpose defense mechanism called DRAM-Locker that can safeguard DRAM against various adversarial DNN weight attacks affecting data or page tables. DRAM-Locker leverages in-DRAM swapping combined with a lock-table to deliver a high level of protection downgrading the performance of targeted weight attacks to a random attack level. Our defense shows reduced latency, extended defense duration, and no reduction in the accuracy of DNNs when applied to various DNN models compared with existing defense mechanisms.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grant No. 2228028, 2216772, and 2216773.

REFERENCES

- [1] Y. Adi *et al.*, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *USENIX*, 2018, pp. 1615–1631.
- [2] A. S. Rakin *et al.*, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, 2019, pp. 1211–1220.
- [3] Y. Kim *et al.*, "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.
- [4] O. Mutlu *et al.*, "Fundamentally understanding and solving rowhammer," in *ASP-DAC*, 2023, pp. 461–468.
- [5] A. Saxena *et al.*, "Pt-guard: Integrity-protected page tables to defend against breakthrough rowhammer attacks," in *DSN*. IEEE, 2023, pp. 95–108.
- [6] Z. Zhang *et al.*, "Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses," in *MICRO*. IEEE, 2020, pp. 28–41.
- [7] P. Frigo *et al.*, "Ttrespass: Exploiting the many sides of target row refresh," in *SP*. IEEE, 2020, pp. 747–762.
- [8] A. Olgun *et al.*, "Dram bender: An extensible and versatile fpga-based infrastructure to easily test state-of-the-art dram chips," *IEEE TCAD*, 2023.
- [9] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA*. IEEE, 2021, pp. 345–358.
- [10] J. S. Kim *et al.*, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *ISCA*. IEEE, 2020, pp. 638–651.
- [11] J. Woo *et al.*, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," *preprint arXiv:2212.12613*, 2022.
- [12] M. Marazzi *et al.*, "Rega: Scalable rowhammer mitigation with refresh-generating activations," in *SP*. IEEE, 2023.
- [13] R. Zhou *et al.*, "Lt-pim: An lut-based processing-in-dram architecture with rowhammer self-tracking," *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 141–144, 2022.
- [14] G. Saileshwar *et al.*, "Randomized row-swap: mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *ASPLOS*, 2022, pp. 1056–1069.
- [15] M. Wi *et al.*, "Shadow: Preventing row hammer in dram with intra-subarray row shuffling," in *HPCA*. IEEE, 2023, pp. 333–346.
- [16] D.-H. Kim *et al.*, "Architectural support for mitigating row hammering in dram memories," *IEEE CAL*, vol. 14, no. 1, pp. 9–12, 2014.
- [17] M. Qureshi *et al.*, "Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *ISCA*, 2022.
- [18] Z. He *et al.*, "Defending and harnessing the bit-flip based adversarial weight attack," in *CVPR*, 2020, pp. 14 095–14 103.
- [19] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *MICRO*. IEEE, 2017, pp. 273–287.
- [20] V. Seshadri, Y. Kim *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *MICRO*, 2013.
- [21] Apple, inc. about the security content of mac efi security update 2015-001. [Online]. Available: <https://support.apple.com/en-au/HT204934>.
- [22] S. M. Seyedzadeh *et al.*, "Counter-based tree structure for row hammering mitigation in dram," *CAL*, vol. 16, 2016.
- [23] E. Lee *et al.*, "Twice: Preventing row-hammering by exploiting time window counters," in *ISCA*, 2019, pp. 385–396.
- [24] Y. Park *et al.*, "Graphene: Strong yet lightweight row hammer protection," in *MICRO*. IEEE, 2020, pp. 1–13.
- [25] A. Kogler *et al.*, "{Half-Double}: Hammering from the next row over," in *USENIX Security*, 2022, pp. 3807–3824.
- [26] Z. Zhang *et al.*, "{SoftTRR}: Protect page tables against rowhammer attacks using software-only target row refresh," in *USENIX*, 2022, pp. 399–414.
- [27] X.-C. Wu *et al.*, "Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells," in *ASPLOS*, 2019, pp. 645–657.
- [28] R. Schilling *et al.*, "Secwalk: Protecting page table walks against fault attacks," in *HOST*. IEEE, 2021, pp. 56–67.
- [29] L. Cojocar *et al.*, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *SP*. IEEE, 2019, pp. 55–71.
- [30] M. Ribeiro *et al.*, "Mlaas: Machine learning as a service," in *ICMLA*. IEEE, 2015, pp. 896–902.
- [31] D. Gruss *et al.*, "Another flip in the wall of rowhammer defenses," in *SP*. IEEE, 2018, pp. 245–261.
- [32] A. Kwong *et al.*, "Rambleed: Reading bits in memory without accessing them," in *SP*. IEEE, 2020, pp. 695–711.
- [33] F. Yao *et al.*, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX*, 2020.
- [34] (2011) Ncsu eda freepdk45. [Online]. Available: <http://www.eda.ncsu.edu/wiki/FreePDK45:Contents>
- [35] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, pp. 1–7, 2011.
- [36] R. Zhou *et al.*, "P-pim: A parallel processing-in-dram framework enabling row hammer protection," in *DATE*. IEEE, 2023.
- [37] A. S. Rakin *et al.*, "Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy," *arXiv preprint arXiv:2103.13813*, 2021.
- [38] J. Li *et al.*, "Defending bit-flip attack through dnn weight reconstruction," in *DAC*. IEEE, 2020, pp. 1–6.