Compromising the Intelligence of Modern DNNs: On the Effectiveness of Targeted RowPress

Ranyang Zhou[†], Jacqueline T. Liu[‡], Sabbir Ahmed[‡], Shaahin Angizi[†], and Adnan Siraj Rakin[‡]
[†]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
rz26@njit.edu, jliu28@binghamton.edu, sahmed9@binghamton.edu, arakin@binghamton.edu, shaahin.angizi@njit.edu

***Abstract*—Recent advancements in side-channel attacks have revealed the vulnerability of modern Deep Neural Networks (DNNs) to malicious adversarial weight attacks. The well-studied RowHammer attack has effectively compromised DNN performance by inducing precise and deterministic bit-flips in the main memory (e.g., DRAM). Similarly, RowPress has emerged as another effective strategy for flipping targeted bits in DRAM. However, the impact of RowPress on deep learning applications has yet to be explored in the existing literature, leaving a fundamental research question unanswered: How does RowPress compare to RowHammer in leveraging bit-flip attacks to compromise DNN performance? This paper is the first to address this question and evaluate the impact of RowPress on DNN applications. We conduct a comparative analysis utilizing a novel DRAM-profile-aware attack designed to capture the distinct bit-flip patterns caused by RowHammer and RowPress. Eleven widely-used DNN architectures trained on different benchmark datasets deployed on a Samsung DRAM chip conclusively demonstrate that they suffer from a drastically more rapid performance degradation under the RowPress attack compared to RowHammer. The difference in the underlying attack mechanism of RowHammer and RowPress also renders existing RowHammer mitigation mechanisms ineffective under RowPress. As a result, RowPress introduces a new vulnerability paradigm for DNN compute platforms and unveils the urgent need for corresponding protective measures.**

I. Introduction

Recent advancements in deep learning have revolutionized applications, including image classification [1], object detection [2], and speech recognition [3]. However, privacy and security concerns surrounding this powerful technology are gaining increasing attention, especially in safety-critical domains like healthcare and finance [4]. Recent attacks exploiting software [5], [6] and system-level [7], [8] vectors demonstrate the feasibility of compromising DNN systems.

Among the various security threats to DNN security, this paper focuses on *adversarial weight attack* [9], [7], [10], [8], [11]. This attack typically injects RowHammer-based faults [12] in DRAM, where model parameters are stored. RowHammer, characterized by repetitive DRAM row activation (i.e., hammering), can significantly degrade DNN performance through precise bit-flip algorithms [9]. Advanced in-DRAM defenses [13], [14], [15], [16] have improved memory system resilience, mitigating RowHammer threats, but similar emerging threats, like RowPress, remain a concern. RowPress shares similarities with RowHammer in exploiting the differential bit states between adjacent rows. The fundamental difference between them lies in the *underlying attack mechanism*. RowPress relies on prolonged row activation, as opposed to RowHammer's frequent and relatively transient row activation, and thus requires considerably fewer activations to induce bit-flips. [17] also shows that these two attack models result in distinct bit-flip patterns, i.e., vulnerable bit profiles. Existing precise bit-flip algorithms such as [9] were designed according to RowHammer. Given the aforementioned similarities and differences between RowHammer and RowPress, we are motivated to leverage and revise such algorithms by incorporating two distinct vulnerable bit profiles to compare and analyze the effect of these two attack models on depleting DNN intelligence. The main contributions of this work are as follows:

(1) We perform RowHammer [18] and RowPress [17] attacks on a Samsung-manufactured DDR4 DRAM chip to profile vulnerable bit-cell locations under both attacks. We then develop a novel profile-aware Bit-Flip Attack (BFA) algorithm that utilizes these two profiles to induce precise targeted bit-flips; (2) We conduct an extensive comparative analysis between RowHammer and RowPress on their effectiveness in depleting DNN intelligence. Eleven DNNs of different sizes and structures trained on three benchmark datasets, including both image and speech data modality, are experimented; and (3) Our results exhibit that, compared to RowHammer, RowPress only requires up to *4*$\times$ fewer bit-flips to undermine DNN performance and can induce up to *20*$\times$ more bit-flips within the same operational duration, making it a stealthier attack with noticeably higher efficacy.

II. Background

DRAM Organization & Commands. A DRAM chip consists of multiple memory banks, each comprising 2D sub-arrays of memory cells arranged in matrices, with billions of cells on modern chips. Each DRAM cell contains a capacitor and an access transistor, where the capacitor's charge state represents a binary '1' or '0' [19]. In idle mode, the memory controller issues a Precharge (`PRE`) command, precharging the Bit-Line (BL) to $\frac{V_{DD}}{2}$. In active mode, the Activate (`ACT`) command activates the Word-Line (WL), allowing DRAM cells to share charge with the BL, altering its voltage. A sense amplifier detects this deviation and amplifies it to V_{DD} or 0, enabling data transfer via read (`RD`) / write (`WR`) commands.

DRAM Timing Parameters. The most basic parameter is the clock cycle (t_{CK}) that is used to measure all parameters. Row Active Time (t_{RAS}) encompasses the temporal window between an `ACT` command and the subsequent `PRE` command, ensuring optimal performance by restoring charge within the DRAM cells on the open row. Row Precharge Time (t_{RP}) signifies the gap between a `PRE` command and the next `ACT` command, closing the open WL and initiating the pre-charging of the BLs to $\frac{V_{DD}}{2}$. Retention time refers to the duration a

978-3-9826741-0-0/DATE25/© 2025 EDAA

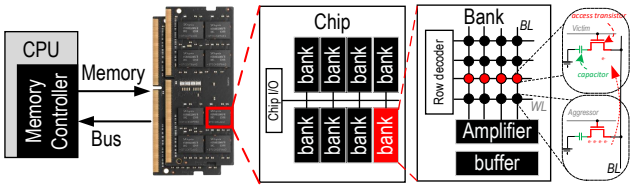

Fig. 1. DRAM organization with fault injection model

memory cell can hold its stored data without needing a refresh, influenced by factors like cell density and electromagnetic interference. These parameters ensure reliable and efficient DRAM operation. In the RowHammer model, the retention time of certain victim rows may significantly reduce. The Refresh Window (t_{REFW}) is the interval within which all DRAM cells must be refreshed to prevent data loss or corruption.

RowHammer in DDR4 & Protection Mechanisms. Kim et al. [12] conducted an extensive study on RowHammer bit-flips in DDR3 modules, finding that approximately 85% of the tested modules were susceptible to RowHammer attacks. Thus, earlier RowHammer research focused on DDR3 systems [20]. DDR4 modules, introduced to create a RowHammer-less landscape, have documented instances of RowHammer on earlier DDR4 generations [21], [22]. TRRespass [23] is the only recently established work exploring the multi-sided fault injection model. Multiple software and hardware mitigation mechanisms reduce RowHammer-based attacks [12], [24], [25]. Hardware-based research is categorized into *victim-focused* mechanisms with probabilistic refreshing (e.g., PRA [15], ProTRR [24]) and *aggressor-focused* mechanisms counting activations (e.g., TRR [26], Hydra [16], CBT [13], Panopticon [27], CRA [15], TWiCe [28], Graphene [29], Mithril [30]). System manufacturers tend to follow mechanisms that detect RowHammer conditions and intervene, such as increasing refresh rates and counter-based approaches. TRR [23] and counter-based detection methods [15], [16], [31] require additional hardware to calculate and record row activations to other fast-read-memory (SRAM [28] / CAM [29]). The controller refreshes the target row if the number reaches the Maximum Activation Count (MAC) [23]. Counter-based solutions add a new DRAM command called Nearby Row Refresh (NRR) [28], [29], issued to refresh the relevant victim rows. JEDEC standards outline three MAC configurations: (1) unlimited, if the DRAM module is RowHammer-free; (2) untested, if the module has not undergone post-production inspection; or (3) T_{MAC}, the specific number of ACTs the module can withstand (e.g., 1M). Most DDR4 modules assert an unlimited MAC value [23].

RowPress. RowPress [17] and RowHammer share the potential for bit-flips in victim rows when bits differ from adjacent (aggressor) rows. Both attacks get worse as DRAM technology scales down to smaller node sizes. The distinction between them lies in how the DRAM rows are accessed (i.e., the failure mechanism): RowHammer repeatedly opens (i.e., activates) and closes an aggressor row many times, whereas RowPress extends the activation duration in the aggressor row, reducing the required number of activations. RowPress-vulnerable cells and RowHammer-vulnerable cells bear $< 0.5\%$ overlap and they exhibit opposite bit-flip directionality trends.

III. Motivation

Existing RowHammer protection mechanisms are designed to monitor the frequency of memory row activations within a specified period. Given that RowPress induces bit-flips in a different manner – prolonging a single activation duration, it is obvious that those RowHammer protection mechanisms will have no effect against RowPress. Although developing a new defense mechanism against RowPress is certainly a meaningful research topic, it is not the focus of this paper; rather, we would like to first conduct a novel systematic comparative analysis of these two fault injection models, focusing on their impact on DNN applications. To be more specific, we focus on answering the following two research questions:

Research Question 1. *How effective is RowPress in depleting the intelligence of deep learning models?*

Research Question 2. *Is it RowHammer or Rowpress that is more effective in performing targeted BFAs to compromise DNN performance?*

We hope that our results will motivate the research community to brainstorm effective defense mechanisms against RowPress.

IV. Threat Model

We adopt a standard practical threat model similar to prior works leveraging remote side-channel attacks to compromise DNN performance [7], [32], [8], [33], [9], [11], [34], [35]. We assume that DNN model inference takes place in a resource-sharing environment as in the Machine-Learning-as-a-Service (MLaaS) [36] setting. The attacker can run user-level unprivileged processes remotely on the same machine where the victim DNN model is deployed. The attacker can map the virtual addresses to physical addresses using several techniques such as leveraging huge page support, hardware-based side-channel attack [22], and memory messaging [37]. The attacker requires knowledge of the DRAM memory addressing scheme, which can be obtained via reverse-engineering [38]. We assume the attacker can engender targeted bit-flips at desired locations using fast and precise multi-bit-flip techniques [7] that ensure the correct hammering patterns for RowHammer or keep a targeted row open for RowPress. Nevertheless, we assume the kernel and operating system are trusted and well-protected [39]. Also following standard practices, we assume ECC does not protect the commercial DRAM and cannot protect large-scale deep learning models against RowHammer [7], [32], i.e., the attacker can bypass the tracking algorithm and prevent the triggering of refresh operations. Finally, we assume a white-box attacker for deep learning models, following earlier works on adversarial weight attacks [9], [8], [7], [11], [34]. Recent remote side-channel attacks have made this white-box threat model assumption practical, as attackers can now effectively obtain critical information such as the number of layers, layer sizes, weight bit sizes, and parameter values through remote side-channel methods [40], [41], [42], [32]. However, the attacker does not have access to training information such as datasets, hyperparameters, etc.

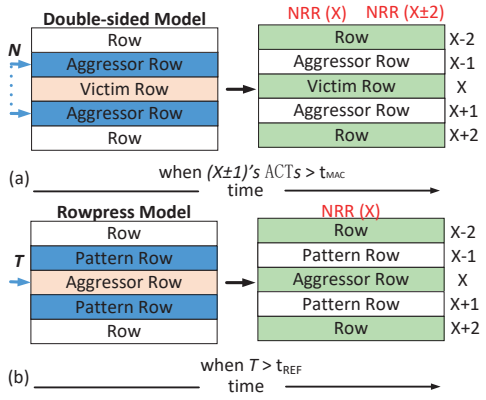

Fig. 2. (a) Double-sided RowHammer attack model, (b) RowPress attack model.

V. Fault Injection Models

In this section, we detail the two fault injection models: RowHammer and RowPress, as well as how we obtain the vulnerable bit-flip profiles.

A. RowHammer

Traditional fault injection models such as double-sided attacks can be effectively defended by counter-based frameworks [14], [15], [16], [31]. As shown in Fig. 2(a), the double-sided RowHammer model mainly affects the victim rows with two aggressor rows X±1. We manually insert data patterns into three rows, assigning all 1s to two aggressors X±1 and all 0s to the victim row X. This is intended to simulate the ideal case where all bits in the victim row differ from those in the aggressor rows. Subsequently, `ACT` command is continuously issued to the aggressor rows. The following testing allows us to establish a range of aggressor rows' HCs, denoted by N, which effectively quantifies the vulnerability level of the victim row. The lower and higher bounds of N correspond to the respective thresholds where the victim row first exhibits bit-flips and where the victim row is entirely reversed due to the attacks. Hence, defense mechanisms will easily identify aggressor rows that have been activated significantly more frequently than other normal rows. As discussed, such defenses establish distinct thresholds depending on the manufacturer of the chips. If the defense mechanism properly detects that the row X±1 reaches the MAC, the NRR will refresh row X and X±2 as shown in Fig. 2(a). Fig. 3(a) shows the timing for such a RowHammer attack. Assuming RowHammer is implemented on row 0x99. F is a flag used to decide whether to issue an NRR command or not. When the Hammer Count (HC) of the row surpasses MAC, which means $t_{RAS} \times HC \geq T_{MAC}$, the memory controller will consider an NRR operation for that row. Algorithm 1 provides a visible function to understand the flow. Firstly, we define data_pattern as all '1's and data_pattern_inv as all '0's in lines 5 and 6, respectively, to create the ideal conditions for a RowHammer attack. Subsequently, in lines 7 and 8, we write these patterns into the aggressor rows and the victim row. Then, the loop operation described from lines 9 to 12 simulates the attacker continuously activating the aggressor rows. Finally, in lines 16 to 18, we transfer the data from the chips to the host PC and check for bit-flips.

Algorithm 1 RowHammer Fault Injection

```
1:  Procedure: RowHammer
2:  Input N
3:  Allocate row_address, bank_address, column_address
4:  Load Data_pattern & Data_pattern_inv
5:  Aggressor_row[row] ← Data_pattern
6:  Victim_row[row] ← Data_pattern_inv     //We assign 0x00000000 to victim rows
7:  For (RowHammer_cnt < N) do
8:      For (row in Row_address) do
9:          ACT Aggressor_row[row];     //Keep hammering Row X ± 1
10:         PRE Aggressor_row[row];
11:         row ← Row_address + 1;
12:     RowHammer_cnt + 1;
13: For (row in Row_address) do
14:     READ Rows[row];
15:     PRE Rows[row];
16:     row ← row + 1;
17: Receive_Data(Platform);     //Write data back to host PC
18: Detect_BitFlips(Victim_Row)
19: end Procedure
```

Common t_{RAS} values for DDR4 memory modules range from around 36 to 48 t_{CK} [43], but these values can differ based on the module's speed rating (e.g., DDR4-2133, DDR4-2400, DDR4-3200, etc.). The duration of a clock cycle for DDR4-2400 memory can be calculated as $t_{CK} = \frac{1}{2400M/s}$. In our design, every t_{RAS} consists of three parts: `ACT`, `Sleep(S)`, and `PRE`, where `Sleep(S)` is set to $5t_{CK}$. Previous research [44] has indicated that the maximum number of HCs typically reaches approximately 1.36 million.

B. RowPress

RowPress [17] aims to bypass the counter-based defense such as CAT [13]. Although most counter-based mechanisms require additional hardware resources to mitigate RowHammer attacks, the threshold for triggering alerts is maintained at an extremely low level. This makes it challenging for attackers to compromise the chips and even generate bit-flips. RowPress extends the open window during activation to prolong charge leakage, thereby generating bit-flips, rather than repeatedly activating rows. Our RowPress implementation is slightly different from [17]. The only difference lies in the attacking target: instead of keeping the aggressor rows open in Fig. 2(a) for a long period, we do so in the victim row in Fig. 2(a). This effectively transforms the victim rows into aggressor rows by directly attacking them. Within this context, we propose to designate the rows adjacent to the aggressor rows as pattern rows and the victim row as the aggressor row, as illustrated in Fig. 2(b). A key similarity shared between RowHammer and RowPress is that bit-flips occur only when the bits in a victim row differ from those in its adjacent rows. The pattern rows serve as victim rows but are utilized to monitor bit-flip occurrences. Here, T, whose value we can customize, represents the open window duration of the victim row. Please note that T cannot exceed the limitation imposed by the refresh time t_{REF}.

Fig. 3(b) shows the timing sequence of a Rowpress attack on row 0x99 as an example. This attack is implemented in such a way that the memory controller does not detect any anomaly because there is always only one activation command involved. Algorithm 2 shows us step-by-step how Rowpress works. It's similar to Algorithm 1, and the main difference lies in line 2, where we take the time T instead of the count N as input. Both

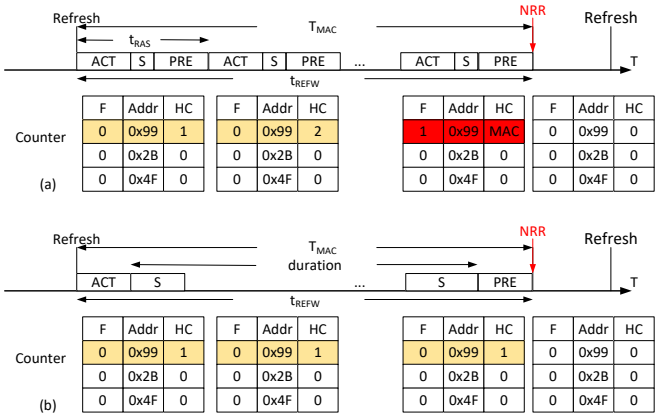

Fig. 3. Timing of (a) RowHammer & (b) RowPress Attack.

algorithms use the same data pattern, but in Algorithm 2, we use one 'ACT' and then wait until we can proceed, as shown in lines 6 to 9. After that, both algorithms do the same things to change and check the data.

VI. DRAM-Profile-aware Attack

The first step for an attacker is to conduct DRAM profiling of the entire chip, leveraging the fault injection model of sections V.A and V.B for RowHammer and RowPress, respectively. After completing the profiling of the DRAM bit-cells under RowHammer and RowPress attacks, a sample DRAM chip will depict a schematic diagram of vulnerable bit-cell similar to Fig. 4. Here, the cross cells are vulnerable to only RowHammer (RH), Black cells are vulnerable to only RowPress (RP), and Dot cells denote vulnerability to both attacks. Next, considering a DRAM chip with N cells to store the deep learning model, we can denote a subset of these cells as C_{rh} and C_{rp}, which indicates the list of cell locations in the DRAM vulnerable to RowHammer and RowPress, respectively. After completing the profiling stage, an attacker will have a set of bit locations C_{rh} or C_{rp}, where the BFA is feasible. A unique page frame number and an offset can identify these locations. The next challenge would be leveraging these bit profiles to formulate an attack objective and search for appropriate vulnerable bit indexes to achieve the desired attack goal for deep learning models.

A. Attack Objective

Our proposed attack algorithm aims to deplete the intelligence of the DNN models. Taking classification as an example task, the attack objective would be to deteriorate the model

Algorithm 2 RowPress Fault Injection

1: **Procedure:** ***CounterBypass***
2: **Input** T
3: *Allocate row_address, bank_address, column_address*
4: *Load Data_pattern & Data_pattern_inv*
5: *Pattern_row[row]* $\leftarrow$ *Data_pattern* //*We assign 0xFFFFFFFF to Pattern rows*
6: *Victim_row[row]* $\leftarrow$ *Data_pattern_inv* //*We assign 0x00000000 to victim rows*
7: *ACT Pattern_row[row];* //*Activate Row X once*
8: *Sleep(T)*
9: *PRE Pattern_row[row];*
10: **For** (*row in Row_address*) **do**
11: *READ Rows[row];*
12: *PRE Rows[row];*
13: *row* $\leftarrow$ *row* + 1;
14: *Receive_Data(Platform);* //*Write data back to host PC*
15: *Detect_BitFlips(Victim_Row)*
16: **end Procedure**

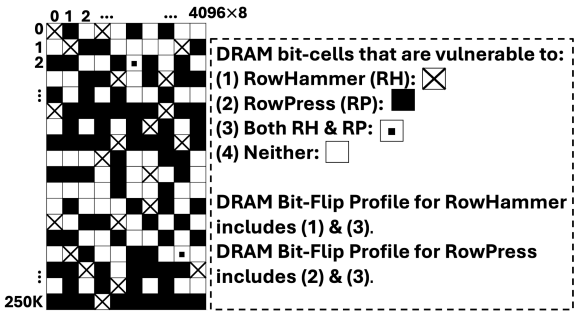

Fig. 4. A schematic diagram of DRAM bit-cells illustrating the huge difference of vulnerable bit-cells under RowHammer (marked with a cross) and RowPress (marked in solid black) attacks in terms of number and location.

accuracy close to a random guess level. For a quantized DNN with a bit precision of n_q-the bit that can be represented in 2's complement binary Tensors as $\{B_l\}_{l=1}^{L}$, where $l \in \{1, 2, ..., L\}$ is the layer index, the attack objective can be formulated as follows:

$$\max_{B} \ \mathcal{L}\Big(f(x; \{\hat{B}_l\}_{l=1}^{L}), y\Big) \\ \sum_{l=1}^{L} \mathcal{D}(\hat{B}_l, B_l) \in \{0, 1, ..., n_b\}, \tag{1}$$

where x and y denote the selected input mini-batch and ground-truth labels; $f(x; \{\hat{B}_l\}_{l=1}^{L})$ compute the outputs of DNN parameterized by $\{\hat{B}_l\}_{l=1}^{L}$; $\hat{B}$ denotes the model weight bits after attack; $\mathcal{L}(\cdot, \cdot)$ computes the cross-entropy loss. The goal is to maximize the loss function $\mathcal{L}$ with the least number of bit-flips (i.e., minimizing $\mathcal{D}$); namely, minimizing attack overhead.

B. Attack Algorithm

An efficient vulnerable bit-search algorithm is needed to attain the above attack objective with minimal bit-flips. We adopt the popular adversarial weight attack algorithm, the BFA algorithm [9], for flipping targeted bits in DNN. BFA is a two-stage search algorithm that utilizes gradient ranking to search for vulnerable weight-bit indices. However, instead of naively applying the BFA algorithm, we incorporate our DRAM bit-flip profiles C_{rh} or C_{rp} appropriately in the search algorithm as shown in Algorithm 3. We introduce a novel search strategy that considers the mapping of neural network weight-bits to bit-cell locations in a hardware DRAM profile. For a quantized DNN represented as $\{B_l\}_{l=1}^{L}$, where $l \in \{1, 2, ..., L\}$ is the layer index; initially, we select a subset of weight-bits, denoted as $\{B_{cl}\}_{l=1}^{L} \in \{B_l\}_{l=1}^{L}$, where $B_{cl} \in B_l \ \forall l$. $\{B_{cl}\}_{l=1}^{L}$ only contains weight-bits that can be mapped to a vulnerable bit-cell location in the DRAM bit-flip profiles: C_{rh} or C_{rp}. In summary, before starting the vulnerable bit-search algorithm, we confine our search within the weight-bits that are mapped to a vulnerable bit-cell location based on the hardware profiles. Note that, as an attacker, we do not alter or control the mapping between model weights and dram cells; rather, we only exploit the vulnerability in the existing mapping. As a result, the attack objective of eqn. 1 can be re-formulated as follows:

$$\max_{B} \ \mathcal{L}\Big(f(x; \{\hat{B}_{cl}\}_{l=1}^{L}), y\Big) \tag{2}$$

Next, our proposed attack performs the conventional BFA search algorithm [9] in an iterative manner. During each iteration, firstly, the intra-layer search identifies the bits with the

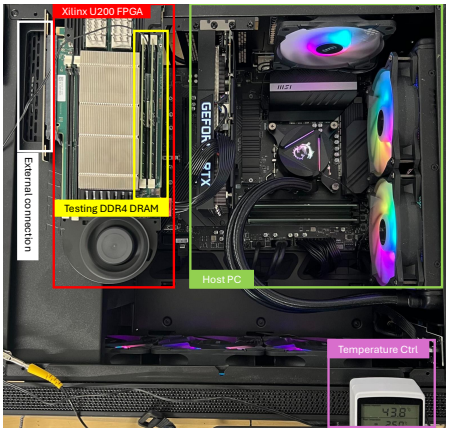

Fig. 5. Our testing infrastructure for DDR4 modules.

highest gradients ($|\nabla_{B_{cl}}\mathcal{L}|$) as vulnerable bit candidates. Then, the inter-layer search compares the bit candidates selected by the intra-layer search by looking at the loss values, i.e., the layer that induces the maximum loss will be elected, and the bits selected in that layer during the intra-layer search will be flipped. The attack continues to the next iteration until the objective is satisfied, as shown in Algorithm 3.

Algorithm 3 DRAM-profile-aware Attack

Procedure: ***DRAM-profile-aware Attack***
Select a set of feasible weight-bits ($\{B_{cl}\}_{l=1}^{L}$) according to the DRAM bit-flip profiles: C_{rh} or C_{rp}.
While *Attack Objective is not Satisfied* **do**
  For (l <= L) **do**
    Find vulnerable weight bit with highest gradient ($|\nabla_{B_{cl}}\mathcal{L}|$)
    Perform bitflip to record loss ($\mathcal{L}\big(f(x;\{\hat{B}_{cl}\}_{l=1}^{L}),y\big)$ *and bit index*
    Restore the bit back
  Enter the layer l_m with maximum loss $\mathcal{L}$
  Perform bitflip at layer l_m on the recorded index
end Procedure

VII. Experiment Results

A. Hardware Setup

DRAM Testing Infrastructure. We extensively modify the DRAM-Bender [45] to have a versatile FPGA-based DRAM attack exploration framework for DDR4 with an in-DRAM compiler API installed on our host machine. Our testing infrastructure (as Fig. 5 illustrates) consists of the Alveo U200 Data Center Accelerator Card [46], which serves as the FPGA that accepts DDR4 modules and runs the test programs based on Algorithms 1 & 2 via sending DDR4 command traces generated by the host machine. The key idea is to take control of memory modules for DDR4 interfaces with straightforward high-level programming to test, characterize, and run the generated programs on the host machine. The driver is designed to send instructions across the PCIe bus to the FPGA to be stored on the board. The temperature is kept below 30°C with INKBIRDPLUS 1800W temperature controller.

Minimizing Interference. To ensure that we directly observe RowHammer and RowPress's circuit-level bit-flips, DRAM refresh [47] and rank-level ECC are disabled. However, proprietary RowHammer protection techniques (e.g., Target Row Refresh [23], [26]) exist.

Tested Commodity DDR4 DRAM Chip. We select one representative Samsung-manufactured DRAM chip with 16GB density (Frequency: 2400MHz, Die revision: B, Org.: x8, Date: 2053) to profile its cell vulnerabilities.

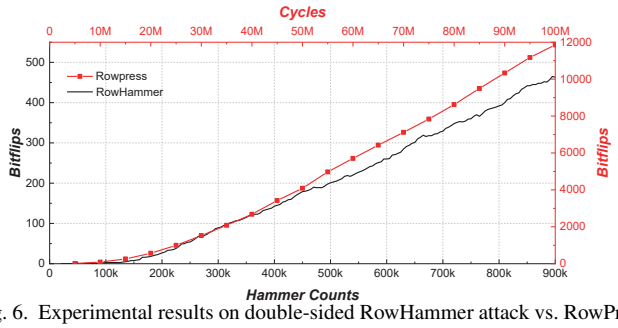

Fig. 6. Experimental results on double-sided RowHammer attack vs. RowPress.

Fair Evaluation Settings for RowHammer & RowPress. When comparing the number of bit-flips, we use HC for RowHammer and the number of cycles elapsing within an activation duration for RowPress, both can be converted into time for a fair comparison. For instance, a 2400MHz DRAM chip with 100 million cycles takes approximately $T = \frac{100M}{2400M} = 41.67ms$. The equivalent HCs in RowHammer can be calculated as $HC = \frac{T}{t_{REF}} \times 1.36M =\sim 885.5K$, with t_{REF} typically being $64ms$ [48].

B. Deep Learning Framework

DNN Architectures & Datasets. We evaluate both RowHammer and RowPress attacks on vision and speech applications. For image classification tasks, we have ResNet-{20,32,44} [49] trained on CIFAR-10 [50], as well as large-scale DNNs trained on ImageNet [51], including ResNet-{34,50,101} [49], DeiT-{T,S,B} [52], and VMamba-T [53]. For speech recognition, we have the M11 model [54] trained on Google's speech command dataset [55]. We perform an 8-bit post-training quantization for all the aforementioned models following [9], [34]. In addition, we run our attack experiments three times to report their average, reducing the impact of random attack initialization (e.g., random test batch selection, and mapping of weights to vulnerable bit-cells).

Evaluation Metric. We report the number of required bit-flips to degrade the model accuracy close to a random guess level (i.e., $\frac{1}{\#\text{output classes}} \times 100\%$). A lower number of bit-flips indicates higher attack efficiency.

C. Results: RowHammer vs. RowPress

1) DRAM Vulnerability Analysis: As shown in Fig. 6, the outcomes of the two distinct attack models are depicted in red and black curves. The red curve, associated with the cycle counts and bit-flips axes on the top and right quantifies the bit-flips induced by RowPress as a function of cycle counts. The black curve, associated with the HCs and bit-flips axes on the bottom and left, demonstrates the correlation of bit-flips with the increasing number of HCs ascribed to the RowHammer attack. This figure indicates that both attack vectors result in an increase in bit-flips over time. Notably, the red curve predominates over the black one for the majority of the observational period, with RowPress producing 20× more bit-flips than RowHammer does.

TABLE I

RESULTS OF ROWHAMMER & ROWPRESS ATTACKS ON DIFFERENT APPLICATIONS, DATASETS, AND DNN ARCHITECTURES. WE REPORT THE NUMBER OF BIT-FLIPS REQUIRED TO DEGRADE THE DNN PERFORMANCE BACK TO A RANDOM GUESS LEVEL.

Dataset	Architecture	#Parameters	Acc. before Attack (%)	Random Guess Acc. (%)	Acc. After RowHammer Attack (%)	#Bit-flips (RowHammer Attack)	Acc. After RowPress Attack (%)	#Bit-flips (RowPress Attack)
CIFAR-10	ResNet-20	0.27M	92.42	10.00	10.39	36	9.14	8
	ResNet-32	0.47M	93.44		10.41	60	10.28	11
	ResNet-44	0.66M	93.90		10.4	53	10.47	14
ImageNet	ResNet-34	21.8M	73.12	0.10	0.14	35	0.13	11
	ResNet-50	25.6M	75.84		0.11	26	0.13	10
	ResNet-101	44.6M	77.20		0.14	30	0.14	11
	DeiT-T	5.7M	71.95		0.15	143	0.12	45
	DeiT-S	22M	79.63		0.15	56	0.07	24
	DeiT-B	86.6M	81.7		0.14	47	0.13	13
	VMamba-T	23M	81.82		0.12	79	0.12	24
Google Speech Command	M11	1.8M	93.2	2.86	2.84	68	2.44	19

Takeaway 1. *Given a similar attack budget (i.e., resources such as time), RowPress produces 20× more bit-flips than RowHammer.*

Echoing [17], this crucial observation suggests that RowPress presents a more pernicious form of attack, capable of inflicting more extensive memory disruption.

2) DRAM-profile-aware Attack Evaluation:

Evaluation of RowPress on Compromising DNNs. Table I summarizes the performance evaluation of our DRAM-profile-aware attack, exhibiting that DNNs of various forms are extremely vulnerable to RowPress. We evaluated ten vision models consisting of three distinct architectural topologies: CNN, vision transformer, and VMamba. CNNs, represented by ResNets, are noticeably more vulnerable than vision transformers and Vmamba, as their intelligence can be depleted with much fewer attack iterations. RowPress also presents effective attack potency on the audio classification model, M11. As shown in the last column of Table I, the RowPress attack requires no more than 45 bit-flips and averages 18 bit-flips to deplete the intelligence of all tested DNN models.

Takeaway 2. *RowPress is highly effective in depleting the intelligence of DNN models of various sizes, topology, training data modality, and resolution.*

RowHammer vs. RowPress Analysis. Fig. 7 presents some representative DNN accuracy degradation curves under our DRAM-profile-aware attack. We observe that the slopes of orange curves (corresponding to the RowPress bit-flip profile) are noticeably steeper than those of blue curves (corresponding to the RowHammer bit-flip profile). We can reasonably attribute this phenomenon to the fact that the DRAM bit-flip profile of RowPress contains more vulnerable bits than that of RowHammer, as Fig. 4 illustrates. *Quantitatively*, the RowPress bit-flip profile contains *more* vulnerable bits; *qualitatively*, bits contained in the RowPress profile are more vulnerable and produce more negative and disastrous effects on the model once they are flipped. This twofold property contributes to RowPress's superior attack efficiency. Both fault injection methods exhibit roughly similar attack efficacy trends across different model sizes and structures. The performance gap is largest for DeiT-B, whereas the gap for VMamba-T is relatively small. Prior works [56], [57] have shown that vision transformers are more resilient against perturbations, which supports our observations.

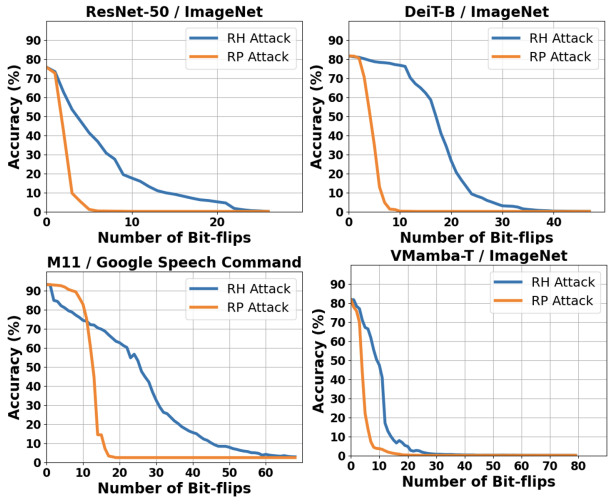

Fig. 7. Accuracy evolution curves versus the number of bit-flips under RowHammer and RowPress attacks.

Takeaway 3. *RowPress exhibits noticeably higher attack efficiency compared to RowHammer, averaging 3.6× fewer bit-flips to attain the same attack objective.*

VIII. CONCLUSION

In this paper, we present a pioneering investigation into the potential impact of RowPress on executing targeted bit-flip attacks against deep neural networks. Our analysis reveals that RowPress enables an attacker to degrade the performance of DNNs with unprecedented efficiency, surpassing the efficacy of the RowHammer attack. We discover that, within the same operational duration, RowPress can induce up to twenty times more bit-flips in a DRAM compared to RowHammer. When a DNN is deployed on the DRAM, to deplete the DNN's intelligence to a random-guess level, our DRAM-profile-aware attack requires up to four times fewer bit-flips with the RowPress bit-flip profile than with the RowHammer bit-flip profile. We have implemented RowPress on physical DRAM chips and rigorously evaluated its impact on the performance of various DNN models. The results confirm that bit-flips executed via RowPress can significantly degrade the performance of all tested models. Therefore, based on our findings, it is crucial for the AI community to address the security threat posed by RowPress by investigating appropriate protective measures.

ACKNOWLEDGMENT

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

REFERENCES

[1] K. He *et al.*, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[2] S. Ahmed *et al.*, "Dfr-tsd: A deep learning based framework for robust traffic sign detection under challenging weather conditions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5150–5162, 2022.

[3] W. Xiong *et al.*, "Achieving human parity in conversational speech recognition," *arXiv preprint arXiv:1610.05256*, 2016.

[4] N. Rajabli *et al.*, "Software verification and validation of safe autonomous cars: A systematic literature review," *IEEE Access*, vol. 9, pp. 4797–4819, 2020.

[5] T. Gu *et al.*, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[6] A. Madry *et al.*, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[7] F. Yao *et al.*, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in {*USENIX*} *Security*, 2020.

[8] S. Hong *et al.*, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks." in *USENIX*, 2019, pp. 497–514.

[9] A. S. Rakin *et al.*, "Bit-flip attack: Crushing neural network with progressive bit search," in *CVPR*, 2019, pp. 1211–1220.

[10] ——, "Tbt: Targeted neural network attack with bit trojan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 198–13 207.

[11] H. Chen *et al.*, "Proflip: Targeted trojan attack with progressive bit flips," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7718–7727.

[12] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ACM SIGARCH Computer Architecture News*, vol. 42. IEEE Press, 2014, pp. 361–372.

[13] S. M. Seyedzadeh *et al.*, "Mitigating wordline crosstalk using adaptive trees of counters," in *ISCA*. IEEE, 2018, pp. 612–623.

[14] S. Saroiu *et al.*, "The price of secrecy: How hiding internal dram topologies hurts rowhammer defenses," in *IRPS*. IEEE, 2022, pp. 2C–3.

[15] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 9–12, 2014.

[16] M. Qureshi *et al.*, "Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *ISCA*, 2022, pp. 699–710.

[17] H. Luo *et al.*, "Rowpress: Amplifying read disturbance in modern dram chips," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–18.

[18] O. Mutlu *et al.*, "Fundamentally understanding and solving rowhammer," in *ASP-DAC*, 2023, pp. 461–468.

[19] R. Zhou *et al.*, "Red-lut: Reconfigurable in-dram luts enabling massive parallel computation," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–8.

[20] M. Seaborn *et al.*, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[21] M. Lipp *et al.*, "Nethammer: Inducing rowhammer faults through network requests," in *EuroS&PW*. IEEE, 2020, pp. 710–719.

[22] D. Gruss *et al.*, "Another flip in the wall of rowhammer defenses," in *SP*. IEEE, 2018, pp. 245–261.

[23] P. Frigo *et al.*, "Trrespass: Exploiting the many sides of target row refresh," in *SP*. IEEE, 2020, pp. 747–762.

[24] M. Marazzi *et al.*, "Protrr: Principled yet optimal in-dram target row refresh," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 735–753.

[25] R. Zhou *et al.*, "Dnn-defender: An in-dram deep neural network defense mechanism for adversarial weight attack," *arXiv preprint arXiv:2305.08034*, 2023.

[26] H. Hassan *et al.*, "Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications," in *MICRO-54*, 2021, pp. 1198–1213.

[27] T. Bennett *et al.*, "Panopticon: A complete in-dram rowhammer mitigation," in *DRAMSec*, vol. 22, 2021, p. 110.

[28] E. Lee *et al.*, "Twice: Preventing row-hammering by exploiting time window counters," in *ISCA*, 2019, pp. 385–396.

[29] Y. Park *et al.*, "Graphene: Strong yet lightweight row hammer protection," in *MICRO*. IEEE, 2020, pp. 1–13.

[30] M. J. Kim *et al.*, "Mithril: Cooperative row hammer protection on commodity dram leveraging managed refresh," in *HPCA*. IEEE, 2022, pp. 1156–1169.

[31] S. M. Seyedzadeh *et al.*, "Counter-based tree structure for row hammering mitigation in dram," *CAL*, vol. 16, 2016.

[32] A. S. Rakin *et al.*, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories," in *IEEE SP*. IEEE, 2022, pp. 1157–1174.

[33] A. Olgun *et al.*, "Quac-trng: High-throughput true random number generation using quadruple row activation in commodity dram chips," in *ISCA*. IEEE, 2021, pp. 944–957.

[34] A. S. Rakin *et al.*, "Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant fpga," in *USENIX Security*, 2021, pp. 1919–1936.

[35] J. Bai *et al.*, "Versatile weight attack via flipping limited bits," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[36] M. Ribeiro *et al.*, "Mlaas: Machine learning as a service," in *ICMLA*. IEEE, 2015, pp. 896–902.

[37] A. Kwong *et al.*, "Rambleed: Reading bits in memory without accessing them," in *SP*. IEEE, 2020, pp. 695–711.

[38] P. Pessl *et al.*, "Drama: Exploiting dram addressing for cross-cpu attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.

[39] R. K. Konoth *et al.*, "Zebram: Comprehensive and compatible software protection against rowhammer attacks," in *USENIX*, 2018, pp. 697–710.

[40] M. Yan *et al.*, "Cache telepathy: Leveraging shared resource attacks to learn dnn architectures," in *29th USENIX Security Symposium*, Aug. 2020, pp. 2003–2020.

[41] Y. Xiang *et al.*, "Open dnn box by power side-channel attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 11, pp. 2717–2721, 2020.

[42] H. Yu *et al.*, "Deepem: Deep neural networks model recovery through em side-channel information leakage," in *HOST*. IEEE, 2020, pp. 209–218.

[43] H. Choi *et al.*, "Reducing dram refresh power consumption by runtime profiling of retention time and dual-row activation," *MICPRO*, vol. 72, 2020.

[44] Z. Lang *et al.*, "Blaster: Characterizing the blast radius of rowhammer," in *3rd Workshop on DRAM Security (DRAMSec) co-located with ISCA 2023*. ETH Zurich, 2023.

[45] A. Olgun *et al.*, "Dram bender: An extensible and versatile fpga-based infrastructure to easily test state-of-the-art dram chips," *TCAD*, 2023.

[46] (2021) Xilinx inc., xilinx alveo u200 fpga board. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo.html

[47] (2020) Jesd79-4c: Ddr4 sdram standard. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo.html

[48] J. Liu *et al.*, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 60–71, 2013.

[49] K. He *et al.*, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015, pp. 1026–1034.

[50] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2010.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[52] H. Touvron *et al.*, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[53] Y. Liu *et al.*, "Vmamba: Visual state space model," *arXiv preprint arXiv:2401.10166*, 2024.

[54] W. Dai, C. Dai *et al.*, "Very deep convolutional neural networks for raw waveforms," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2017, pp. 421–425.

[55] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[56] K. Mahmood *et al.*, "On the robustness of vision transformers to adversarial examples," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 7838–7847.

[57] J. Zhang *et al.*, "Transferable adversarial attacks on vision transformers with token gradient regularization," in *CVPR*, 2023, pp. 16 415–16 424.