PuDHammer: Experimental Analysis of Read Disturbance Effects of Processing-using-DRAM in Real DRAM Chips

İsmail Emir Yüksel Akash Sood Ataberk Olgun Oğuzhan Canpolat Haocong Luo
F. Nisa Bostancı Mohammad Sadrosadati A. Giray Yağlıkçı Onur Mutlu

ETH Zürich

Processing-using-DRAM (PuD) is a promising paradigm for alleviating the data movement bottleneck using a DRAM array's massive internal parallelism and bandwidth to execute very wide data-parallel operations. Performing a PuD operation involves activating multiple DRAM rows in quick succession or simultaneously, i.e., multiple-row activation. Multiple-row activation is fundamentally different from conventional memory access patterns that activate one DRAM row at a time. However, repeatedly activating even one *DRAM row (e.g., RowHammer) can induce bitflips in unaccessed DRAM rows because modern DRAM is subject to read disturbance, a worsening safety, security, and reliability issue. Unfortunately, no prior work investigates the effects of multiple-row activation, as commonly used by PuD operations, on DRAM read disturbance.*

In this paper, we present the first characterization study of read disturbance effects of multiple-row activation-based PuD (which we call PuDHammer) using 316 real DDR4 DRAM chips from four major DRAM manufacturers. Our detailed characterization results covering various operational conditions and parameters (i.e., temperature, data patterns, access patterns, timing parameters, and spatial variation) show that 1) PuDHammer significantly exacerbates the read disturbance vulnerability, causing up to 158.58× reduction in the minimum hammer count required to induce the first bitflip (HC_{first}), compared to RowHammer, 2) PuDHammer is affected by various operational conditions and parameters, 3) combining RowHammer with PuDHammer is more effective than using RowHammer alone to induce read disturbance errors (e.g., compared to RowHammer, doing so reduces HC_{first} by 1.66× on average across all tested rows), and 4) PuDHammer bypasses an in-DRAM RowHammer mitigation mechanism called Target Row Refresh and induces more bitflips than RowHammer.

To develop future robust PuD-enabled systems in the presence of PuDHammer, we 1) develop three countermeasures and 2) adapt and evaluate the effectiveness of state-of-the-art RowHammer mitigation standardized by industry, called Per Row Activation Counting (PRAC). We show that the adapted PRAC incurs large performance overheads to mitigate PuDHammer (e.g., an average performance overhead of 48.26% across 60 five-core multiprogrammed workloads). We hope and expect that our findings motivate and guide system-level and architectural solutions to enable read-disturbance-resilient future PuD systems.

1. Introduction

Modern computing systems move vast amounts of data between main memory (DRAM) and processing elements (e.g., CPU and GPU) [1, 2]. Unfortunately, this data movement is a major bottleneck that consumes a large fraction of execution time and energy in many modern applications [1–28]. Processing-using-DRAM (PuD) [29–34] is a promising paradigm that can alleviate the data movement bottleneck. PuD uses the analog operational properties of the DRAM array circuitry to enable massively parallel in-DRAM computation (i.e., PuD operations), which can be used to accelerate important applications including databases and web search [29, 30, 32, 35–43], data analytics [29, 44–48], graph processing [32, 48–51], genome analysis [52–57], cryptography [58, 59], hyper-dimensional computing [60–62], and generative AI [63–72].

A wide variety of PuD operations (e.g., in-DRAM data copy and bulk bitwise operations) rely on a key PuD technique called *multiple-row activation*, which accesses (activates) multiple DRAM rows in quick succession or simultaneously [29–32, 40, 73–84]. Multiple-row activation is fundamentally different from conventional DRAM operations that access only a *single* DRAM row at a time.

Unfortunately, with aggressive technology node scaling, repeatedly accessing even a single DRAM row disturbs the data integrity of *unaccessed* physically-adjacent DRAM rows and causes bitflips [85–152]. *RowHammer* [93] and *RowPress* [153] are two prominent examples of DRAM read disturbance phenomena where a DRAM row (i.e., victim row) can experience bitflips when a nearby DRAM row (i.e., aggressor row) is 1) repeatedly activated (i.e., hammered) [34, 93, 116, 152] or 2) kept open for a long period (i.e., pressed) [34, 145, 153, 154]. Unfortunately, no prior work explores the read disturbance effects of multiple-row activation-based PuD operations.

In this paper, we present the first experimental characterization of the read disturbance effects of *multiple-row activation-based PuD operations*, PuDHammer, on 316 commercial off-the-shelf (COTS) DDR4 DRAM chips from four major DRAM manufacturers (in 40 DRAM modules). We characterize the read disturbance effect of two types of multiple-row activation: 1) consecutive multiple-row activation (which we call CoMRA), used for in-DRAM data copy [40, 73, 74, 77, 79, 83, 155] (§4) and 2) simultaneous multiple-row activation (which we call SiMRA), used for in-DRAM bitwise operations [29, 30, 73, 74, 76, 78, 79, 81, 82, 155] (§5) under various operational conditions and parameters (i.e., data patterns, temperature, access patterns, timing parameters, and spatial variation). We also analyze 1) how combining RowHammer with PuDHammer

(i.e., a combined pattern that performs RowHammer and PuDHammer repeatedly) affects read disturbance (§6) and 2) the effectiveness of an in-DRAM mitigation mechanism, broadly referred to as Target Row Refresh (TRR) [119, 125, 156–158], against PuDHammer (§7).

Based on our characterization, we make 26 new empirical observations and share 9 key takeaway lessons. We highlight four of our major new results. First, repeatedly performing multiple-row activation greatly increases the DRAM chip's read disturbance vulnerability. We find that both CoMRA and SiMRA decrease the minimum hammer count required to induce the first bitflip (HC_{first}) in all tested DRAM chips from four manufacturers. We observe that the lowest HC_{first} observed due to CoMRA and SiMRA are 13.98× and 158.58× lower than the lowest HC_{first} observed due to RowHammer, respectively. Second, operational conditions and parameters impact PuDHammer (especially SiMRA). We find that hammering with SiMRA is significantly affected by data pattern and row on time (i.e., RowPress [153]) and can change the average HC_{first} of victim rows in a DRAM chip by up to 57.80× and 270.27× across all tested data patterns and row on time values. Third, a combined RowHammer and PuDHammer access pattern is much more effective at inducing read disturbance bitflips than using RowHammer alone. We observe that, compared to RowHammer, the average HC_{first} of victim rows in a DRAM chip decreases by up to 1.34×, 1.22× when RowHammer is combined with CoMRA or SiMRA, respectively. We find that a combination of RowHammer, CoMRA, and SiMRA is the most effective access pattern among the tested patterns and reduces average HC_{first} of victim rows in a DRAM chip by 1.66×. Fourth, we observe that in a tested SK Hynix DDR4 DRAM module, both CoMRA and SiMRA bypass the TRR mechanism and induce more bitflips than RowHammer. For example, SiMRA and CoMRA respectively induce 11340× and 1.10× more bitflips than RowHammer on average in the presence of TRR.

Our characterization results suggest that future PuD-enabled systems should take PuDHammer into account to maintain the fundamental robustness property of memory isolation. Based on our findings, to mitigate PuDHammer, we 1) develop and qualitatively analyze three countermeasures against PuDHammer that modify the DRAM chips and DRAM interface (§8.1) and 2) adapt and evaluate the state-of-the-art RowHammer solution standardized by industry, called Per Row Activation Counting (PRAC) [159–163] (§8.2). We find that the adapted PRAC solution to PuDHammer incurs an average system performance overhead of 48.26% across all tested 60 five-core multiprogrammed workload mixes.

This paper makes the following key contributions:

- To our knowledge, this is the first work to analyze and experimentally demonstrate the interaction between read disturbance and Processing-using-DRAM (PuD) operations in COTS DRAM chips. We extensively characterize the read disturbance effect of multiple-row activation-based PuD operations on 316 chips from 40 real DRAM modules.
- Our results show that 1) multiple-row activation greatly amplifies the DRAM read disturbance errors across all tested manufacturers (e.g., up to 158.58× reduction in HC_{first}), 2) the read disturbance effect of multiple-row activation depends on operational conditions and parameters with large variations in some cases (e.g., up to 270.27× change in HC_{first}), 3) combining RowHammer with multiple-row activation is more effective than using RowHammer alone for inducing the first read disturbance error (e.g., 1.66× reduction in HC_{first} on average), and 4) PuDHammer bypasses the in-DRAM TRR mechanism, inducing many more bitflips in a DRAM row than RowHammer does, in the presence of TRR (e.g., 11340× more bitflips on average).
- We develop and analyze four potential ways to mitigate PuDHammer. We adapt and evaluate the effectiveness of the industry's state-of-the-art RowHammer mitigation, PRAC, and show that adapted PRAC incurs large performance overheads to mitigate PuDHammer.
- Our takeaway lessons (from both characterization and mitigation) call for future work on understanding the underlying device-level causes of PuDHammer bitflips and other innovative solutions to mitigate PuDHammer bitflips to enable read-disturbance-resilient future PuD systems.

2. Background

2.1. Dynamic Random Access Memory (DRAM)

DRAM Organization. Fig. 1 shows the hierarchical organization of a modern DRAM-based main memory. The memory controller connects to a DRAM module over a memory channel. A module contains one or multiple DRAM ranks that time-share the memory channel. A rank consists of multiple DRAM chips. Each DRAM chip has multiple DRAM banks, each containing multiple subarrays.

Within a subarray, DRAM cells form a two-dimensional structure interconnected over *bitlines* and *wordlines*. The row decoder in a subarray decodes the row address and drives one wordline out of many. A row of DRAM cells on the same wordline is referred to as a DRAM *row*. The DRAM cells in the same *column* are connected to the sense amplifier via a bitline. A DRAM cell stores a binary data value in the form of electrical charge on a capacitor (V_{DD} or 0 V), and this data is accessed through an access transistor, driven by the wordline to connect the cell capacitor to the bitline.

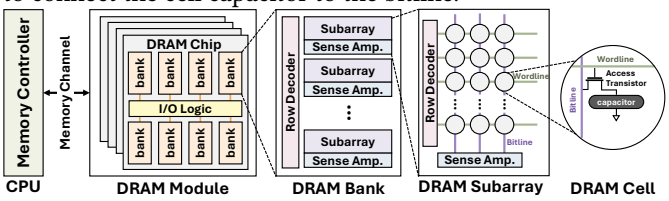

Figure 1: Hierarchical organization of modern DRAM.

DRAM Access. The memorycontroller serves memory access requests by issuing DRAM commands, e.g., row activation (`ACT`), bank precharge (`PRE`), data read (`RD`), data write (`WR`), and refresh (REF) while respecting certain timing parameters to guarantee correct operation [21, 164–174]. The memory controller issues an `ACT` command alongside the bank address and

row address corresponding to the memory request's address to activate a DRAM row. During the row activation process, a DRAM cell shares its charge, and thus, its initial charge level needs to be restored (via a process called *charge restoration*). The latency from the start of a row activation until the completion of the DRAM cell's charge restoration is called t_{RAS}. To access another row in an already activated DRAM bank, the memory controller must issue a `PRE` command to close the opened row and prepare the bank for a new activation. The minimum latency between issuing a `PRE` command and opening a row with an `ACT` command is called t_{RP}.

DRAM Refresh. A DRAM cell is inherently leaky and thus loses its stored electrical charge over time [175–177]. To maintain data integrity, a DRAM cell is periodically refreshed with a time interval called the t_{REFW}, which is typically $64\,\text{ms}$ (e.g., [167, 168, 177, 178]) at normal operating temperature (i.e., up to $85\,^{\circ}\text{C}$). To refresh all cells in a timely manner, the memory controller periodically issues a refresh (`REF`) command with a time interval called the t_{REFI}, which is typically $7.8\,\mu\text{s}$ (e.g., [125, 167, 168, 178]) or $3.9\,\mu\text{s}$ (e.g., [164, 165, 169]) at normal operating temperature.

2.2. DRAM Read Disturbance

Read disturbance is the phenomenon that reading data from a memory or storage device causes physical disturbance (e.g., voltage deviation, electron injection, electron trapping) on another piece of data that is *not* accessed but physically located nearby the accessed data. Two prime examples of read disturbance in modern DRAM chips [108, 116, 152, 179] are RowHammer [93], and RowPress [153], where repeatedly accessing (hammering) or keeping active (pressing) a DRAM row induces bitflips in physically nearby DRAM rows. In RowHammer and RowPress terminology, the row that is hammered or pressed is called the *aggressor* row, and the row that experiences bitflips the *victim* row. For read disturbance bitflips to occur, 1) the aggressor row needs to be activated more than a certain threshold value, defined as the minimum hammer count required to induce the first bitflip (HC_{first}) [144] and/or 2) the aggressor row needs to be open for a long period of time (i.e., $t_{\text{AggON}} > t_{\text{RAS}}$) [153].

2.3. Processing-using-DRAM (PuD)

Processing-using-DRAM (PuD) is an emerging paradigm that can alleviate the bottleneck caused by frequent data movement between processing elements (e.g., CPU) and main memory [29–34, 40, 48, 73–78, 80, 80–83, 155, 180–194]. PuD enables massively parallel in-DRAM computation by leveraging intrinsic analog operational properties of the DRAM circuitry. Many PuD works [29–31, 33, 34, 40, 73–75, 77–79, 81–83, 155] enables 1) in-DRAM data copy & initialization by leveraging consecutive multiple-row activation (which we call CoMRA) and simultaneous multiple-row activation (which we call SiMRA) and 2) in-DRAM bitwise operations by leveraging SiMRA.

In-DRAM Data Copy & Initialization. RowClone [40] enables data movement within a subarray at a row granularity by modifying DRAM circuitry. RowClone alleviates the energy and execution time costs of transferring data between the DRAM and the processing units. Prior works [73, 77] experimentally demonstrate that the RowClone operation can be performed in COTS DRAM chips by enabling *consecutive activation of two rows* in the same subarray, which we call CoMRA. A recent work [79] demonstrates that commercial off-the-shelf (COTS) DRAM chips can copy one source row to up to 31 different destination rows by simultaneously activating up to 32 rows in the same subarray.

In-DRAM Bitwise Operations. Prior works [29, 30] demonstrates that 1) simultaneously activating three DRAM rows leads to the computation of the bitwise MAJority function (and thus the AND and OR functions) on the contents of the three rows due to the charge sharing principles that govern the operation of the shared bitlines and sense amplifiers, 2) bitwise NOT of a row can be performed through the sense amplifier, with modifications to DRAM circuitry. Many operations envisioned by these works [29, 30] can *already* be performed in *real unmodified* COTS DRAM chips, by violating manufacturer-recommended DRAM timing parameters [73–75, 78, 79]. Recent works show that COTS DRAM chips can perform 1) the bitwise MAJ operation with up to nine inputs (i.e., MAJ3, MAJ5, MAJ7, and MAJ9) by simultaneously activating multiple rows in the same subarray [73–75, 79] and 2) up to 16-input AND, NAND, OR, NOR operations, and NOT operation by simultaneously activating multiple rows in two neighboring subarrays [78].

2.4. Motivation

PuD is a promising paradigm that has the potential to reduce or eliminate costly data movement between processing elements and main memory [29–34, 40, 48, 73–78, 80, 80–83, 155, 180–194]. Many PuD techniques (§2.3) leverage an analog DRAM operation called multiple-row activation, where multiple DRAM rows are activated simultaneously or in quick succession to perform in-DRAM processing [29–33, 40, 73–83, 155, 194, 195]. Multiple-row activation fundamentally differs from standard DRAM accesses, where a single row is activated at a time (§2.1). This fundamental difference could result in significant implications for future PuD-enabled systems, as repeatedly activating *even* a single DRAM row can induce bitflips in other unaccessed DRAM rows due to DRAM read disturbance phenomena (§2.2). Unfortunately, no prior work explores the read disturbance effects of multiple-row activation. Our goal in this paper is to close this gap. We aim to empirically understand, characterize, and provide insights into the interaction between read disturbance and multiple-row activation.

3. Methodology

We describe our COTS DRAM chip testing infrastructure (§3.1) and the COTS DDR4 chips tested for our characterization study (§3.2). We explain the methodology of our different characterization experiments in their corresponding sections, §4, §5, and §6.

3.1. COTS DRAM Testing Infrastructure

We conduct COTS DRAM chip experiments using DRAM Bender [75, 196] (built upon SoftMC [197, 198]), an FPGA-based

DDR4 testing infrastructure that provides precise control of DDR4 commands. Fig. 2 shows our experimental setup that consists of four main components: 1) a host machine that generates the test program and collects results, 2) an FPGA development board [199], programmed with DRAM Bender, 3) a thermocouple temperature sensor and heater pads pressed against the DRAM chips to maintain a target temperature level, and 4) a temperature controller [200] that keeps the temperature at the desired level.

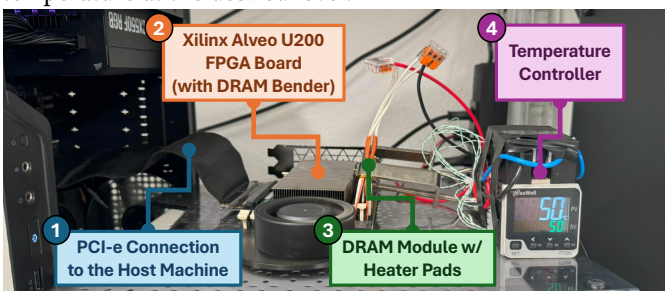

Figure 2: Our DRAM Bender [75] based experimental setup.

Eliminating Interference Sources. To observe read disturbance bitflips at the circuit level, we eliminate potential sources of interference, by taking three measures, similar to the methodology used by prior works [125, 144–146, 153]. First, we disable periodic refresh during the execution of test programs to prevent potential on-DRAM-die TRR mechanisms [119, 125] from refreshing victim rows so that we can observe the DRAM chip's behavior at the circuit-level. Second, we strictly bound the execution time of test programs within the refresh window of the DRAM chips to avoid data retention failures interfering with read disturbance failures. Third, we verify that the modules and chips have neither rank-level nor on-die ECC [201, 202]. With these measures, we directly observe and analyze all bitflips without interference.

3.2. COTS DDR4 DRAM Chips Tested

Table 1 provides the 316 (40) COTS DDR4 DRAM chips (modules) along with the chip manufacturer (Chip Mfr.), number of modules (#Modules), number of chips (#Chips), die revision (Die Rev.), chip density (Density), and chip organization (Org.).

Table 1: Summary of DDR4 DRAM chips tested.

Chip Mfr.	#Modules	#Chips	Die Rev.	Density	Org.
SK Hynix	1	8	A	4Gb	x8
	8	64	A	8Gb	x8
	2	16	C	16Gb	x8
	6	48	D	8Gb	x8
Micron	1	8	B	4Gb	x8
	4	32	E	16Gb	x16
	4	32	F	16Gb	x8
	2	16	R	8Gb	x8
Samsung	1	8	A	16Gb	x8
	5	40	B	16Gb	x8
	1	4	C	4Gb	x16
	1	8	C	16Gb	x8
	1	8	E	4Gb	x8
Nanya	3	24	C	8Gb	x8

Logical-to-Physical Row Mapping. DRAM manufacturers use mapping schemes to translate logical (memory-controller-visible) addresses to physical row addresses [21, 89, 93, 109, 119, 120, 177, 201, 203–210]. To account for in-DRAM row address mapping, we reverse engineer the physical row address layout in all chips, following the prior works' methodology [144–146, 153].

4. Read Disturbance Effect of Consecutive Multiple-Row Activation (CoMRA) in COTS DRAM Chips

We demonstrate the read disturbance effect of consecutive multiple row activation CoMRA in COTS DRAM chips. CoMRA is used to perform in-DRAM data copy operations in real DRAM chips [73–79]. We repeatedly perform consecutive activation of two rows (source row and destination row) to copy a source row's content into a destination row. This section describes our key idea to hammer with CoMRA (§4.1), our experimental methodology for understanding the read disturbance vulnerability caused by the CoMRA operation (§4.2), and presents our COTS DRAM chip characterization results (§4.3).

4.1. Hammering with CoMRA

Key Idea. We exploit the key property of CoMRA to induce read disturbance bitflips in COTS DRAM chips: activating two rows consecutively in quick succession to repeatedly perform in-DRAM data copy operations. To do so, we repeatedly perform consecutive activation of a source row (src) and a destination row (dst) in the same subarray to induce read disturbance bitflips in neighboring rows of the src and dst. In this scenario, src and dst are the aggressor rows, and their neighboring rows are victim rows. Fig. 3 illustrates our key idea to hammer with CoMRA. Depending on the location of source and destination rows, we can either craft 1) a double-sided attack where src and dst are sandwiching a victim row (Fig. 3a) or 2) a single-sided attack where src and dst are far away (Fig. 3b).

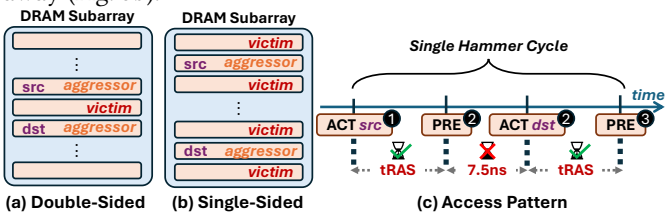

Figure 3: Example of a double-sided CoMRA attack (a), a single-sided CoMRA attack (b), and the access pattern of double-sided and single-sided CoMRA (c).

Access Pattern & Operation. Fig. 3c shows both single-sided and double-sided access patterns of hammering with CoMRA. Our attack consists of three key steps. First, we issue ACT (ACTIVATE) src to activate the src row (❶) and wait for t_{RAS}, which ensures the sense amplifier senses the data in the src row and restores the charges of cells. Second, we issue PRE (PRECHARGE) and ACT dst back-to-back by violating the t_{RP} timing parameter to activate the dst while bitlines and sense amplifiers still have src's content [73, 77, 79] (❷). Doing so copies the src row's data to the dst row [75–79, 79, 211]. Third, we wait for t_{RAS} to ensure that the in-DRAM copy operation completes and issue a PRE to prepare the bank for

the next attack access (❸). We call this three-step in-DRAM copy procedure one hammer cycle. To induce read disturbance bitflips in the victim rows, we repeatedly perform CoMRA.

4.2. Experimental Methodology

DRAM Subarray Boundaries. Understanding the read disturbance effect of the in-DRAM copy operation in COTS DRAM chips requires reverse engineering DRAM subarray boundaries as the in-DRAM copy operation works only if the source and destination rows are located in the same subarray [40, 73, 76–79, 83, 211]. We repeatedly perform the RowClone operation for *every* possible source and destination row address in each tested bank. When we observe that the destination row gets the same content as the source row after the in-DRAM data copy, we conclude that the source row and the destination row are in the same subarray. Based on this observation, we reverse engineer the subarray boundaries and determine which rows are in the same subarray.

Read Disturbance Vulnerability Metric. To characterize a DRAM module's vulnerability to read disturbance, we examine the minimum hammer count required to induce the first bitflip (HC_{first}), where we count the hammer cycles (i.e., each pair of activations to the src and dst rows as one hammer). A lower HC_{first} indicates a higher vulnerability to read disturbance.

***HC_{first}* Algorithm.** For every tested parameter we evaluate (e.g., data pattern and temperature), we find the HC_{first} for each tested victim row using the bisection-method algorithm used by prior works [145, 146, 153]. We terminate the HC_{first} search when the difference between the current and previous HC_{first} measurements is no larger than 1% of the previous measurements. For every tested row, we repeat the HC_{first} search five times and report the minimum HC_{first} value we observe.

Victim Row Location in the Subarray. To understand the effects of spatial variation on read disturbance, we analyze how the location of a victim row in a subarray affects the read disturbance vulnerability. We categorize a victim row's location in a subarray into five regions: 1) "Beginning": the first 20% rows in the subarray (e.g., the first 100 rows in a subarray with 500 rows, row 0 to row 99), 2) "Beginning-Middle": the second 20% rows in the subarray (e.g., row 100 to row 199), "Middle": the third 20% rows in the subarray (e.g., row 200 to row 299), "Middle-End": the fourth 20% rows in the subarray (e.g., row 300 to row 399), and 5) "End": the last 20% rows in the subarray (e.g., row 400 to row 499).

Data Pattern. We use the four data patterns (`0x00`, `0xFF`, `0xAA`, and `0x55`) that are widely used in memory reliability testing [212, 213] and by prior work on DRAM characterization (e.g., [93, 144, 145, 153, 154, 211, 214–216]). We fill aggressor rows (src and dst) with these data patterns while initializing victim rows with the negated data pattern (e.g., if aggressor rows are `0x00`, victim rows are `0xFF`). For each DRAM row, we define the worst-case data pattern (*WCDP*) as the data pattern that causes the lowest HC_{first}. All experiments are conducted using WCDP unless stated otherwise.

Temperature. We perform our experiments at four temperature levels: 50°C, 60°C, 70°C, and 80°C. All experiments are conducted at 80°C unless stated otherwise.

Timing Delay. We sweep the timing delay between `PRE`→`ACT` dst command presented in Fig. 3c. All experiments are conducted with a timing delay of violated $7.5ns$ (as in Fig. 3c) unless stated otherwise.

Number of Instances Tested. To maintain a reasonable testing time, we select six subarrays in a bank per DRAM module: two subarrays from the beginning of the bank, two subarrays from the middle of the bank, and two subarrays from the end of the bank. Within each subarray, we test all rows.

4.3. COTS DRAM Chip Characterization

This section presents our characterization of the read disturbance caused by consecutive activation of two rows in COTS DRAM chips.

Double-Sided CoMRA vs. RowHammer. We investigate the variation in HC_{first} across rows when we perform double-sided CoMRA and double-sided RowHammer. In Fig. 4, the left plot shows the distribution of the change in HC_{first} (in percentage) when we perform double-sided CoMRA compared to double-sided RowHammer for all tested chips from four manufacturers. The x-axis represents the percentage of all victim rows from all tested chips from all tested manufacturers, sorted from the most positive HC_{first} change (i.e., CoMRA has higher HC_{first} compared to RowHammer) to the most negative HC_{first} change (i.e., CoMRA has lower HC_{first} compared to RowHammer). In Fig. 4, the right plot shows the lowest HC_{first} observed across all tested 316 DRAM chips from four manufacturers for double-sided CoMRA and RowHammer.

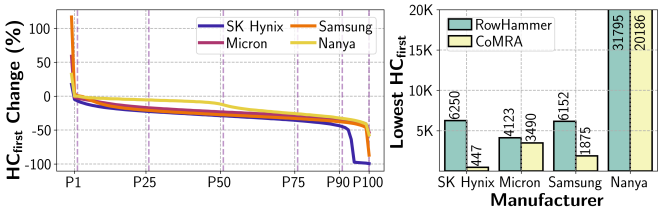

Figure 4: Distribution of the change in HC_{first} with double-sided CoMRA compared to double-sided RowHammer (left) and the lowest HC_{first} observed with double-sided CoMRA and RowHammer (right) for each manufacturer.

Observation 1. Hammering with double-sided CoMRA significantly decreases HC_{first} compared to double-sided RowHammer.

We observe that when rows are hammered with double-sided CoMRA, the lowest HC_{first} observed is 447, 3490, 1875, and 20186 for SK Hynix, Micron, Samsung, and Nanya chips, respectively. Compared to double-sided RowHammer, double-sided CoMRA decreases the lowest HC_{first} by 13.98×, 1.18×, 3.28×, and 1.58× for SK Hynix, Micron, Samsung, and Nanya chips, respectively.

We hypothesize that the reason for double-sided CoMRA's lower HC_{first} compared to double-sided RowHammer is that the reduced interval between closing of the src wordline and

the activation of the dst wordline enhances trap-assisted electron migration from near the neighboring aggressor wordline towards the victim node [142, 217] (i.e., trapped electron density is higher when the neighboring aggressor wordline is *just* closed). We call for future research to fundamentally understand CoMRA's read disturbance.

Observation 2. Hammering with double-sided CoMRA decreases HC_{first} for a large fraction of DRAM rows.

In the left plot of Fig. 4, we observe that compared to double-sided RowHammer, 99% of DRAM rows experience the first bitflip with fewer activation counts when performing double-sided CoMRA for all four manufacturers.

Takeaway 1. CoMRA exacerbates DRAM's vulnerability to read disturbance in all four major manufacturers.

Data Pattern. We analyze the effect of data pattern on HC_{first} across DRAM rows when we perform double-sided CoMRA. Fig. 5 shows the HC_{first} distribution across all tested DRAM rows (y-axis) for four data patterns (x-axis).[1]

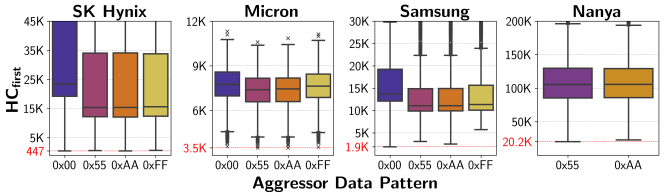

Figure 5: HC_{first} distribution of double-sided CoMRA with different aggressor data patterns. Victim rows have negated aggressor data pattern.

Observation 3. CheckerBoard pattern (i.e., 0x55/0xAA) is, in general, the most effective data pattern among the ones tested.

When performing double-sided CoMRA, in most cases, 0x55/0xAA is the most effective at inducing bitflips compared to 0x00/0xFF, similar to RowHammer [93, 145, 153]. For example, in Samsung chips, average HC_{first} for 0x55 is 17346, whereas for 0x00 is 21423. However, in some cases, we also observe that the worst-case data pattern is not the CheckerBoard pattern, similar to prior works [93, 144, 145, 153, 216].

Temperature. Fig. 6 shows the HC_{first} distribution of hammering with double-sided CoMRA at four different temperature levels: 50°C, 60°C, 70°C, and 80°C. Each subplot is dedicated to a different manufacturer, where the x-axis shows the tested temperature levels.

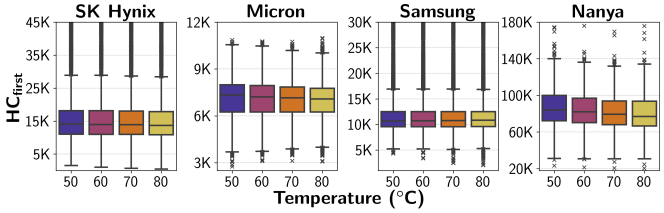

Figure 6: HC_{first} distribution of hammering with double-sided CoMRA at different DRAM chip temperatures.

[1]Due to the complicated true/anti cell pattern of Nanya chips, we could not observe bitflips within a refresh window with 0xFF and 0x00 data patterns.

Observation 4. Read disturbance effects of hammering with double-sided CoMRA tend to get worse as temperature increases.

We observe that when we perform double-sided CoMRA, as temperature increases from 50°C to 80°C, the lowest HC_{first} decreases by 3.45×, 2.13×, and 1.14× for SK Hynix, Samsung, and Nanya, respectively. On the other hand, for Micron, the effect is the opposite, where the lowest HC_{first} increases as the temperature increases (e.g., by 1.14× from 50°C to 80°C). A prior work [145] hypothesizes that the relation between read disturbance vulnerability and temperature is caused by the nonmonotonic behavior of charge-trapping characteristics of DRAM cells, which results in individual DRAM rows exhibiting different behavior. This hypothesis could also explain Micron's HC_{first} trend as we sweep temperature.

Takeaway 2. Hammering with CoMRA is affected by temperature and data pattern. Worst-case data pattern and temperature tend to differ for individual DRAM rows.

Single-Sided CoMRA vs. RowHammer. We analyze the HC_{first} variation across DRAM rows for three techniques: 1) single-sided CoMRA where src and dst are far away from each other (e.g., 100 rows apart), 2) single-sided RowHammer where we keep hammering one aggressor row, and 3) far double-sided RowHammer where the access pattern is the same as single-sided CoMRA except we use t_{RP} latency between PRE to ACT dst. We evaluate the far double-sided RowHammer to observe the effect of reduced PRE to ACT dst latency in single-sided CoMRA. Fig. 7 shows the HC_{first} distribution of these three techniques, where each subplot is dedicated to a different manufacturer and colored boxes present different techniques. We highlight the lowest observed HC_{first} of each technique for every manufacturer with a yellow rectangle.

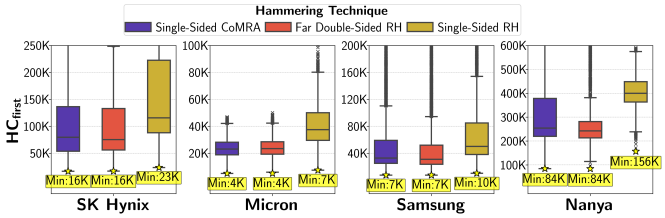

Figure 7: HC_{first} of single-sided CoMRA and RowHammer.

Observation 5. Single-sided CoMRA decreases HC_{first} compared to single-sided RowHammer and exhibits similar HC_{first} distribution with far double-sided RowHammer.

We observe that, for all four manufacturers, single-sided CoMRA 1) is more effective than single-sided RowHammer, and 2) performs similarly to far double-sided RowHammer. For example, in SK Hynix chips, the lowest observed HC_{first} for single-sided CoMRA is 16495, which is 1.42× and 1.02× lower than single-sided RowHammer and far double-sided RowHammer, respectively.

We hypothesize that this observation is caused by increased delay to issue ACT after PRE to an aggressor row (i.e., t_{AggOFF} [153]). For example, in the single-sided CoMRA and

far double-sided RowHammer access patterns (ACT src-PRE-ACT dst), the frequency of hammering with src is relatively lower than single-sided RowHammer (ACT src-PRE) due to activating dst in-between every activation of src. Prior work [153] shows that increasing t_{AggOFF} single-sided RowHammer reduces the HC_{first}. As a result, single-sided CoMRA and far double-sided RowHammer 1) exhibit similar HC_{first} distributions, and 2) are more effective than single-sided RowHammer.

CoMRA vs. RowPress. To understand the read disturbance effect of CoMRA better, we analyze how increasing the time that an aggressor row stays active (t_{AggOn}) (i.e., increasing the latency of ACT $dst\rightarrow$ PRE) affects HC_{first} in double-sided CoMRA and compare double-sided CoMRA against double-sided RowPress. Fig. 8 shows the HC_{first} distribution across DRAM rows for four t_{AggOn} values: 36ns (t_{RAS}, the nominal timing parameter), 144ns ($4\times t_{RAS}$), 7.8μs (t_{REFI}), and 70.2μs ($9\times t_{REFI}$).

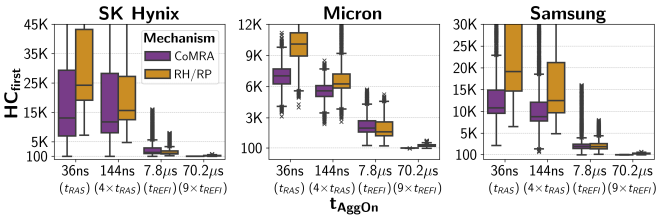

Figure 8: HC_{first} distribution of double-sided CoMRA and RowPress with different t_{AggOn} values.

Observation 6. When performing double-sided CoMRA, increasing t_{AggOn} significantly reduces HC_{first}.

For example, in Micron chips, CoMRA with t_{AggOn}=70.2μs leads to a 78.74× reduction in average HC_{first} compared to CoMRA with t_{AggOn}=36ns. We also observe a similar trend for RowHammer/RowPress: when the t_{AggOn} increased to 70.2μs from 36ns, average HC_{first} decreases by 31.15×, similar to prior works [153, 211, 218].

Observation 7. At t_{AggOn}=t_{REFI}, double-sided RowPress becomes more effective than double-sided CoMRA.

We observe that double-sided CoMRA leads to a reduction in HC_{first} compared to double-sided RowHammer/RowPress accross t_{AggOn} values of 36ns, 144ns, and 70.2μs. For example, at a t_{AggOn} value of 144ns for Micron chips, average HC_{first} of CoMRA is 1.27× lower than RowPress. However, at 7.8μs, RowPress becomes more effective and has 1.17× lower average HC_{first} than CoMRA.

Takeaway 3. *Pressing* with CoMRA is more effective than hammering with CoMRA.

Timing Delay. We analyze the violated latency for PRE→ACT dst to develop more insights into the read disturbance effect of CoMRA. Fig. 9 shows the HC_{first} distribution of double-sided CoMRA for four violated PRE→ACT dst latency values.

Observation 8. HC_{first} increases as the latency of PRE→ACT dst increases.

For example, as the PRE→ACT dst latency increases from

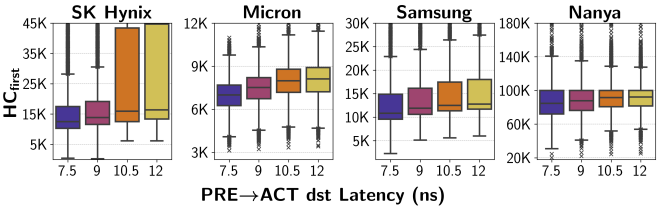

Figure 9: HC_{first} distribution of double-sided CoMRA for varying numbers of latency.

7.5ns to 12ns, average HC_{first} increases by 3.10×, 1.18×, 1.17×, and 3.01× for SK Hynix, Micron, Samsung, and Nanya. We hypothesize that the CoMRA access pattern becomes more of a RowHammer access pattern as the latency increases. As a result, HC_{first} distribution exhibits higher values when the latency increases.

Copy Direction. We analyze how copy direction affects the HC_{first}. Instead of copying from src to dst, we copy from dst to src and test the HC_{first} change distribution. Fig. 10 shows the change in HC_{first} across rows.

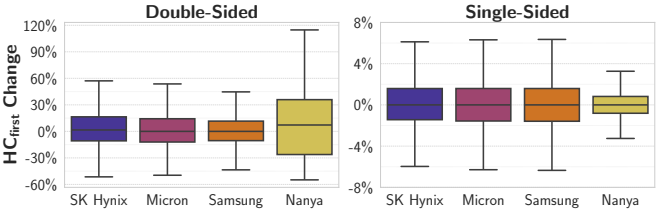

Figure 10: Distribution of the change in HC_{first} when the copy direction is reversed.

Observation 9. In most cases, copy direction has a small effect on HC_{first}.

We observe that, across all four manufacturers, the average HC_{first} change is 2.79% (i.e., 1.03×) and 0.40% (i.e., 1.004×) for double-sided and single-sided, respectively. However, in a small fraction of DRAM rows, we observe up to 20.10× and 2.39× change in HC_{first} for double-sided and single-sided, respectively.

Spatial Variation. Fig. 11 shows the HC_{first} distribution of double-sided CoMRA across DRAM rows (y-axis) based on a victim row's location in a subarray (x-axis).

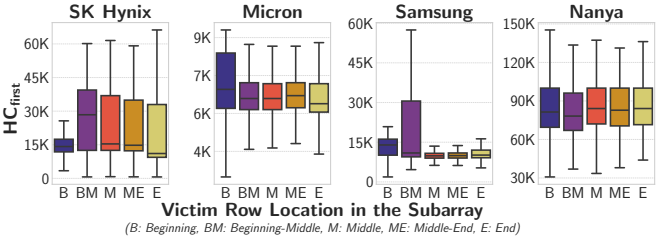

Figure 11: HC_{first} of double-sided CoMRA based on the victim row's location in a subarray.

Observation 10. HC_{first} varies based on the victim row's location in a subarray.

We observe that the physical location of the victim row in the subarray can lead to variations in average HC_{first} of up to 1.40× for SK Hynix, 2.25× for Micron chips, 2.57× for

Samsung chips, and 1.04× for Nanya chips.

Observation 11. Each manufacturer has a different HC_{first} variation trend with the victim row location in the subarray.

For example, in SK Hynix chips, victim rows at the beginning of a subarray (i.e., the first 20% rows) exhibit lower average HC_{first} than others, while in Samsung chips, victim rows in the middle have the lowest average HC_{first}. We hypothesize that differences in DRAM circuit design and manufacturing process technology could lead to a different HC_{first} variation trend for each manufacturer.

Takeaway 4. CoMRA 1) decreases HC_{first} significantly compared to RowHammer, and 2) gets affected by data pattern, temperature, timing delays, copy direction, and spatial variation.

5. Read Disturbance Effect of Simultaneous Multiple-Row Activation (SiMRA) in COTS DRAM Chips

We present an experimental analysis of SiMRA's read disturbance effect in COTS DRAM chips. SiMRA can be used to perform many in-DRAM operations, including 1) bulk bitwise operations (e.g., AND, OR,NAND, and NOR) [73–75, 78, 79], 2) copying data to multiple rows [79], and 3) generating true random numbers [76]. To understand the read disturbance effect of SiMRA, we repeatedly perform SiMRA to hammer multiple rows simultaneously.

5.1. Hammering with SiMRA

Key Idea. We exploit SiMRA to *simultaneously* activate multiple rows and repeatedly hammer many rows. To do so, we issue `ACT-PRE-ACT` command sequence in quick succession, similar to prior works [73–76, 78, 79]. This results in simultaneously activated rows to be aggressor rows and their neighboring rows to be victim rows. Fig. 12 illustrates our key idea to perform hammering using SiMRA. Depending on the location of simultaneously activated rows, we can perform either 1) a double-sided attack where any two activated rows sandwich a victim row (R_a and R_b in Fig. 12a) and 2) a single-sided attack where activated rows do *not* sandwich a victim row (Fig. 12b).

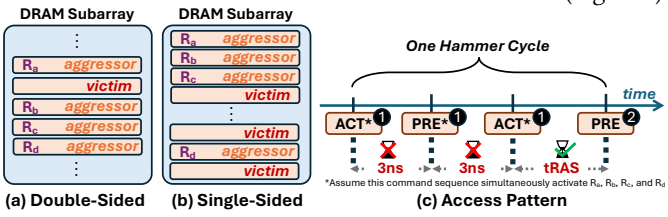

Figure 12: Example of (a) double-sided SiMRA attack, (b) single-sided SiMRA attack, and (c) their access pattern.

Access Pattern & Operation. Fig. 12c illustrates our access pattern to hammer using SiMRA. Our attack consists of two key steps. First, we issue an `ACT-PRE-ACT` command sequence with reduced timing parameters to simultaneously activate multiple aggressor rows, which are R_a, R_b, R_c, and R_d in our example (❶). Second, we wait for t_{RAS} and issue a `PRE` command to complete the SiMRA operation (❷). We count one SiMRA operation as a single hammer and repeatedly perform SiMRA operations to induce bitflips in victim rows.

5.2. Experimental Methodology

We use the same metrics, algorithms, data patterns, and temperature levels as CoMRA experiments (§4.3). This section explains the rest of the methodology unique to SiMRA experiments.

Finding Simultaneously Activated Rows. Prior works [76, 78, 79] show that issuing an `ACT-PRE-ACT` command sequence and following with a `WR` command overwrites the simultaneously activated rows with data supplied with the `WR` command. We follow the same methodology and reverse engineer the simultaneously activated rows with the `ACT-PRE-ACT` command sequence for every row address in a tested subarray. Similar to prior works [78, 79], we observe that COTS DRAM chips can activate 2, 4, 8, 16, and 32 rows in the same subarray. We define a term called *SiMRA-N* where N is the number of simultaneously activated rows (i.e., 2, 4, 8, 16, or 32). For example, SiMRA-16 stands for simultaneous activation of 16 rows.

Timing Delay. We sweep two key timing delays in the `ACT-PRE-ACT` command sequence in Fig. 12c: 1) from `ACT` to `PRE` and 2) from `PRE` to `ACT`. All experiments are conducted with timing delays of $3ns$ (as shown in Fig. 12c) unless stated otherwise.

Number of Instances Tested. To maintain a reasonable testing time, we select a total of six subarrays in one bank per DRAM module: two subarrays from the beginning of the bank, two subarrays from the middle of the bank, and two subarrays from the end of the bank. Within each subarray, we randomly test 100 different groups of rows that are simultaneously activated each for 2-, 4-, 8-, 16-, and 32-row activation.

5.3. COTS DRAM Chip Characterization

This section presents our characterization of the read disturbance effect of SiMRA in SK Hynix chips. While we test all four manufacturers, we note that we do *not* observe SiMRA in Samsung, Micron, and Nanya chips, similar to prior works [74, 76, 78, 79, 195].[2]

Double-Sided SiMRA vs. RowHammer. We investigate the variation in HC_{first} across DRAM rows when we perform double-sided SiMRA and double-sided RowHammer. In Fig. 13, the left plot shows the distribution of the change in HC_{first} (in percentage) when we perform double-sided SiMRA with varying numbers of simultaneously activated rows compared to double-sided RowHammer.[3] The x-axis represents the percentage of all tested victim rows, sorted from the most positive HC_{first} change to the most negative HC_{first} change. Fig. 13 (right) shows the lowest HC_{first} observed across all DRAM chips for double-sided SiMRA with varying numbers of simultaneously activated rows and RowHammer.

Observation 12. Hammering with double-sided SiMRA greatly decreases HC_{first}.

[2]Prior works [74,76,78,79,195] hypothesize that some chips ignore a DRAM command when the command greatly violates nominal timing parameters.

[3]Even though we simultaneously activate 32 rows, we could not find an activated row group that sandwiches a victim row. Hence, for double-sided SiMRA, we show up to 16-row activation (i.e., SiMRA-16).

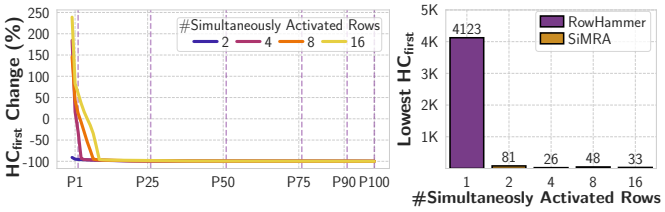

Figure 13: Distribution of the change in HC_{first} change with double-sided SiMRA compared to double-sided RowHammer (left) and the lowest HC_{first} observed with double-sided SiMRA and RowHammer (right).

For double-sided SiMRA with 2-, 4-, 8- and 16-row activation, respectively 100.00%, 98.79%, 97.40%, and 94.94%, of victim rows experience lower HC_{first} than RowHammer. At least 25.19% of victim rows exhibit more than 99% reduction in HC_{first}. We observe that performing double-sided SiMRA decreases the HC_{first} down to 26.

We observe that the reduction in HC_{first} is *not* proportional to the number of simultaneously activated rows since at least 25.19% of victim rows exhibit >99% reduction (i.e., >100×reduction) in HC_{first} for *all* tested N (i.e., numbers of simultaneously activated rows). For example, one tested victim row shows a 158.58x (124.94x) reduction in HC_{first} when performing double-sided SiMRA with 4-row activation (32-row activation), which is significantly higher than 4x (32x) and non-monotonic with N.

Takeaway 5. SiMRA drastically exacerbates DRAM's vulnerability to read disturbance.

Data Pattern. We analyze the effect of data pattern on HC_{first} across DRAM rows when we perform double-sided SiMRA. Fig. 14 shows the HC_{first} distribution of double-sided SiMRA for four tested data patterns. Each boxplot is dedicated to a different number of simultaneously activated rows (i.e., N).

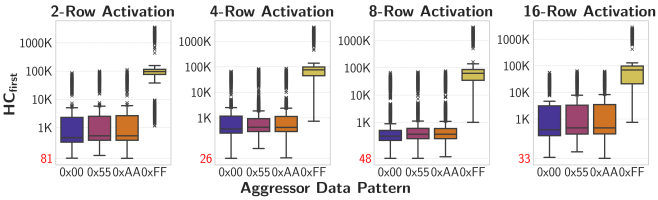

Figure 14: HC_{first} distribution of double-sided SiMRA for different aggressor data patterns and numbers of activated rows. Victim rows have negated aggressor data pattern.

Observation 13. Data pattern significantly affects HC_{first}.

We observe that, across all tested N values, initializing the victim rows with a 0x00 data pattern increases average HC_{first} by up to 57.80× when compared to other data patterns.

Observation 14. SiMRA and RowHammer have opposite bitflip directions.

The dominant bitflip direction for SiMRA is 1 to 0 in all tested N values as also shown in Observation 13 that 0xFF as the victim data pattern (i.e., 0x00 as aggressor data pattern) results in much lower HC_{first} than 0x00 (i.e., 0xFF as aggressor data pattern). For RowHammer (not shown in the figure), we observe that the dominant bitflip direction is 0 to 1, similar to prior work's findings [153, 154, 219].

Takeaway 6. Double-sided SiMRA is significantly affected by data pattern. The directionality of SiMRA and RowHammer bitflips are opposite.

Temperature. Fig. 15 shows the HC_{first} of double-sided SiMRA-N at four temperature levels: 50°C, 60°C, 70°C, and 80°C.

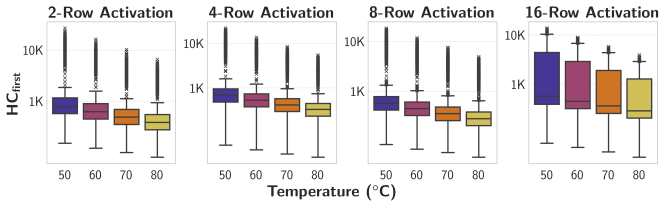

Figure 15: HC_{first} distribution of double-sided SiMRA at different temperatures and numbers of simultaneously activated rows.

Observation 15. HC_{first} decreases as temperature increases.

Increasing temperature consistently decreases HC_{first} for all N. For example, from 50°C to 80°C, average HC_{first} decreases by 3.24×, 3.10×, 3.02×, and 3.26× for 2-, 4-, 8-, and 16-row activation.

We hypothesize that double-sided SiMRA has a different underlying silicon-level mechanism compared to double-sided RowHammer due to two observations. First, SiMRA and RowHammer have opposite bitflip directionality. Prior works on both device-level [217, 220] and real DRAM characterization [153, 154, 219] show that the dominant bitflip direction for RowHammer is 1 to 0, whereas we observe that for SiMRA, it is 0 to 1 (Observation 14). Second, SiMRA and RowHammer have different temperature dependence. Prior real DRAM characterization works [145, 153] show that there is no clear relation between RowHammer and temperature, whereas we observe that for SiMRA, increasing temperature worsens the read disturbance vulnerability (Observation 15). We hope and expect that future device-level studies (inspired by this work) will develop a rigorous device-level understanding of the read disturbance effect of CoMRA and SiMRA operations as device-level studies (e.g., [221, 222]) did for RowPress after the RowPress paper [153] demonstrated the empirical basis for the RowPress phenomenon.

Single-Sided SiMRA. Fig. 16 shows the HC_{first} distribution for single-sided SiMRA-N and single-sided RowHammer. The lowest HC_{first} of each technique is highlighted with a yellow rectangle.

Observation 16. Single-sided SiMRA exhibits a lower average & minimum HC_{first} than single-sided RowHammer.

For example, compared to single-sided RowHammer, the lowest HC_{first} decreases by 1.17× when performing single-sided SiMRA with 32 rows (i.e., 32-row activation).

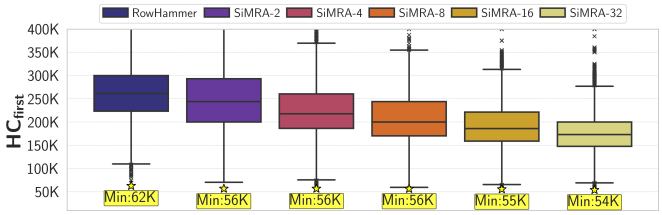

Figure 16: HC_{first} distribution of single-sided SiMRA with varying numbers of activated rows and RowHammer.

Observation 17. HC_{first} consistently decreases as the number of simultaneously activated rows increases when performing single-sided SiMRA.

For example, average (lowest) HC_{first} for SiMRA-32 is 1.47× (1.05×) lower than SiMRA-2. We hypothesize that this potentially results from having more aggressor rows beyond the immediate aggressor of the victim row (i.e., far-aggressor rows) which contribute to inducing bitflips in the victim row, thereby reducing HC_{first}, similarly to the findings of prior work [128].

SiMRA vs. RowPress. To develop a better understanding of SiMRA's read disturbance effect, we conduct an experiment where we sweep t_{AggOn} for double-sided SiMRA. We test three t_{AggOn} values in addition to t_{RAS} (four in total) after issuing ACT-PRE-ACT commands, which keeps the simultaneously activated rows open for longer times. Fig. 17 shows the HC_{first} distribution of double-sided SiMRA and RowPress (we refer to RowPress at 36ns or t_{RAS} as RowHammer).

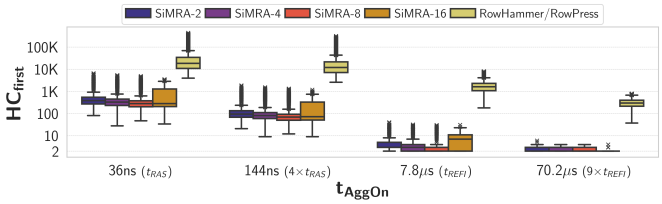

Figure 17: HC_{first} distribution of RowPress and SiMRA with varying numbers of row activations and t_{AggOn} values.

Observation 18. HC_{first} decreases greatly as t_{AggOn} increases.

For example, when t_{AggOn} increases from 36ns to 70.2μs with SiMRA, average HC_{first} decreases by between 144.93×-270.27× across all numbers of simultaneously activated rows.

Timing Delay. Fig. 18 shows how HC_{first} changes when we sweep timing delays between ACT-PRE and PRE-ACT in ACT-PRE-ACT.

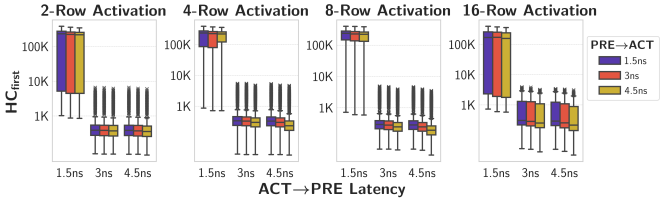

Figure 18: HC_{first} distribution of double-sided SiMRA for different timing delays between ACT-PRE and PRE-ACT.

Observation 19. Increasing PRE-ACT latency slightly decreases HC_{first}.

For example, SiMRA-16 with ACT→PRE=3ns, average HC_{first} decreases by 1.23× when PRE→ACT increases from 1.5ns to 4.5ns.

Observation 20. Specific latency values fail to fully activate aggressor rows, which leads to increase in HC_{first}.

Choosing ACT→PRE latency as 1.5ns, increases average HC_{first} by 2.28×. We observe that (similar to prior work [79]) some aggressor rows are not fully activated (i.e., not all cells are activated) in very low latencies. We hypothesize that due to this partial activation of aggressor rows leads to a drastic increase in HC_{first} in some cases.

Spatial Variation. Fig. 19 shows the HC_{first} distribution of double-sided SiMRA (y-axis) with varying numbers of activated rows (each subplot) based on a victim row's location in a subarray (x-axis).

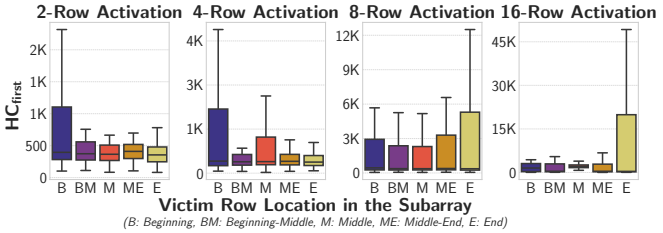

Figure 19: HC_{first} distribution of double-sided SiMRA based on the victim row's location in a subarray.

Observation 21. The effectiveness of SiMRA depends on the victim row's location in a subarray.

HC_{first} varies across different victim row locations in a subarray, and this variation is different for each number of simultaneously activated rows. For example, for 4-row activation, victim rows at the beginning of a subarray experience the highest HC_{first} distribution, whereas for 8-row activation, victim rows at the end of a subarray exhibit the highest HC_{first} distribution.

Takeaway 7. The operating parameters (i.e., data pattern, temperature, timing delays, and spatial variation) impact SiMRA's read disturbance effect.

6. Combined Read Disturbance Effect of RowHammer with PuD in COTS DRAM Chips

We demonstrate how combining RowHammer with CoMRA and SiMRA affects read disturbance vulnerability in real DRAM chips. To understand how much the observed HC_{first} using RowHammer decreases when combined with multiple-row activation: 1) we hammer a victim row using multiple-row activation for a fixed number of times and 2) we perform RowHammer.[4]

6.1. Experimental Methodology

Fig. 20 shows how RowHammer can be combined with a one (Fig. 20a presents an example with CoMRA) or two (Fig. 20b) multiple-row activation techniques. We combine RowHammer

[4]Our tested access pattern is one of many ways of combining the CoMRA, SiMRA, and RowHammer access patterns. There could be more effective access patterns that reduce HC_{first} even more. We leave this exploration, which requires extensive characterization and analysis, to future work.

with a single multiple-row activation technique in two steps. First, we characterize the HC_{first} of a row when only hammered with a multiple-row activation technique (e.g., Fig. 20a-❶ for CoMRA). Second, we characterize the HC_{first} of a victim row with only RowHammer (e.g., Fig. 20a-❷) Third, we hammer the victim row to a fraction of the multiple-row activation's HC_{first} value. We use three levels of hammer count for our experiments: 10%, 50%, and 90% of multiple-row activation's HC_{first} value. For example, if a victim row has HC_{first} of A when performing CoMRA only (Fig. 20-❶) and if we are to test for 10%, we hammer that victim row for A/10 times with CoMRA, as in Fig. 20a. Fourth, we continue hammering the victim row with RowHammer until the first bitflip is observed (e.g., Fig. 20a-❸). We combine RowHammer with both multiple-row activation techniques similarly by characterizing HC_{first} values (Fig. 20b-❶) and hammering a victim row with each technique up to a fraction and then continuing with RowHammer until a bitflip is observed (Fig. 20b-❸). Fifth, we report the HC_{first} change of these combined access patterns compared to using RowHammer only (e.g., B-C decrease in Fig. 20a and C-D decrease in Fig. 20b).

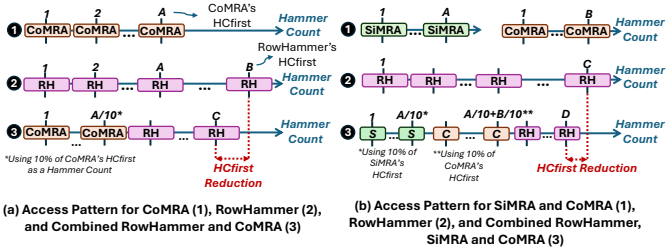

Figure 20: Example access pattern of combining RowHammer (a) with CoMRA and (b) CoMRA and SiMRA together.

6.2. COTS DRAM Chip Characterization

We present our characterization of the combined RowHammer and multiple-row activation pattern in the same chips that are used in SiMRA characterization (§5). We conduct our experiments at 80°C using the double-sided pattern for all techniques with WCDP for each DRAM row.

Combining RowHammer with CoMRA. Fig. 21 shows the change in HC_{first} distribution and the absolute HC_{first} values across DRAM rows when we perform combined double-sided RowHammer and CoMRA, compared to double-sided RowHammer. The x-axis shows the hammer count for CoMRA as a percentage of HC_{first} observed for each row.

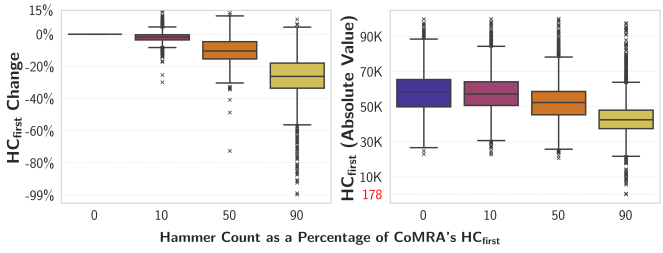

Figure 21: Change in HC_{first} (left) and absolute HC_{first} values (right) when combining RowHammer and CoMRA. Hammer count of 0% represents performing RowHammer only.

Observation 22. DRAM rows experience significantly lower HC_{first} when performing combined RowHammer and CoMRA.

Compared to double-sided RowHammer, 95.33% of tested victim rows show lower HC_{first} when we combine double-sided RowHammer with CoMRA. HC_{first} reduction increases as we increase the hammering fraction of CoMRA. For example, when a victim row is first hammered with CoMRA until 90% and 10% of CoMRA's HC_{first} and then hammered with RowHammer, HC_{first} decreases by 1.34× and 1.02× (compared to RowHammer), respectively.

Combining RowHammer with SiMRA. Fig. 22 shows the change in HC_{first} and the absolute HC_{first} values when we perform combined double-sided RowHammer and SiMRA.

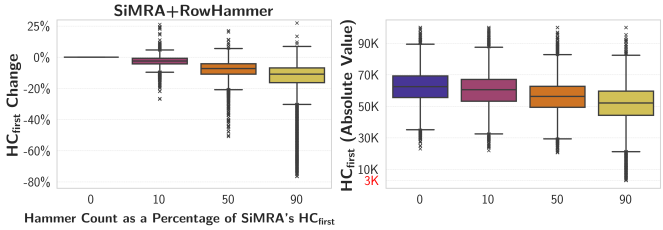

Figure 22: Change in HC_{first} (left) and absolute HC_{first} values (right) when combining RowHammer and SiMRA. Hammer count of 0% represents performing RowHammer only.

Observation 23. DRAM rows tend to experience lower HC_{first} when performing combined RowHammer and SiMRA.

Combining RowHammer with SiMRA 1) decreases HC_{first} as the hammer count for SiMRA increases and 2) is less effective than combining RowHammer with CoMRA. For example, in combined RowHammer and SiMRA, the average HC_{first} change in the highest hammer count (i.e., 90% percentage) is 1.22× lower than combined RowHammer and CoMRA. We hypothesize that the most vulnerable cell to RowHammer in some victim rows is not vulnerable to SiMRA. Thus, combining RowHammer with SiMRA does not decrease the HC_{first} of all tested victim rows as much as combining RowHammer with CoMRA.

Combining RowHammer with CoMRA and SiMRA. Fig. 23 shows the change in HC_{first} when we combine double-sided RowHammer with CoMRA and SiMRA together (e.g., Fig. 20b-❸).

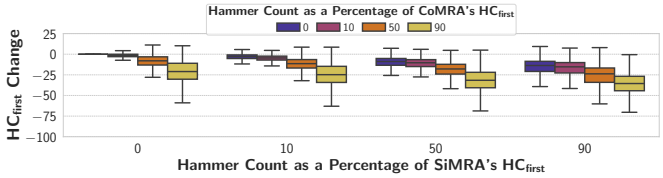

Figure 23: Change in HC_{first} for combined RowHammer, CoMRA, and SiMRA compared to only RowHammer.

Observation 24. Combined RowHammer with CoMRA and SiMRA together is the most effective combined access pattern to decrease HC_{first}.

We observe that when we combine RowHammer with

CoMRA and SiMRA, the majority of the DRAM rows exhibit much lower HC_{first} compared to performing only RowHammer. The minimum average HC_{first} of combined RowHammer with CoMRA and SiMRA together is 1.66× lower than performing only RowHammer.

Takeaway 8. Combining RowHammer with CoMRA and/or SiMRA decreases HC_{first}, and combining RowHammer with CoMRA and SiMRA together is the most effective pattern across all tested access patterns.

7. PuDHammer in the Presence of In-DRAM TRR

To prevent read disturbance bitflips, DRAM manufacturers equip their chips with a mitigation mechanism broadly referred to as Target Row Refresh (TRR) [119, 125, 156–158]. Manufacturers do *not* disclose the operational principles or implementations of proprietary TRR versions (e.g., in DDR4 [119, 125]). At a high level, TRR 1) identifies potential aggressor rows as the memory controller issues activate commands to the DRAM chip and 2) preventively refreshes victim rows when the memory controller issues a periodic REF command. We demonstrate that in a tested SK Hynix DDR4 DRAM module [223], both CoMRA and SiMRA bypass the TRR mechanism and induce more read disturbance bitflips than RowHammer.

Uncovering the TRR Mechanism. We uncover the TRR mechanism in the tested module using U-TRR [125, 224]. We observe that the tested module uses a sampling-based TRR mechanism, where TRR probabilistically identifies one aggressor row by sampling row addresses of the last 450 ACT commands before issuing the TRR-capable REF (i.e., REF command that refreshes victim rows).

Access Pattern. We use the custom access pattern reported by U-TRR [125] for RowHammer and CoMRA. The custom access pattern (which we call the N-sided pattern) uses N aggressor rows, (we sweep N from 1 to 10) and one dummy row. Our custom access pattern consists of four key steps. First, we perform a total of 156 hammers for N aggressor rows in one periodic refresh window (see §2.1).[5] Second, we hammer the dummy row 468 times (i.e., 156×3 times, the maximum number of ACT commands that can be issued in three refresh windows) to make the TRR mechanism refresh the victims of the dummy row while activations to the N aggressor rows go unnoticed. For SiMRA, instead of using N aggressor rows, we perform SiMRA (described in §5) that simultaneously activates multiple (2, 4, 8, 16, and 32) rows. We hammer each aggressor row 500K times. We repeat this test 5 times and report the average, maximum, and minimum bitflip counts across all tests.

Results. Fig. 24 shows the number of bitflips observed in victim rows on average across all tests, with error bars representing the range across all tests. The top plot shows the results when TRR is disabled (i.e., w/o TRR), and the bottom subplot shows the results when TRR is enabled (i.e., w/ TRR). Each column of subplots is dedicated to a different hammering technique.

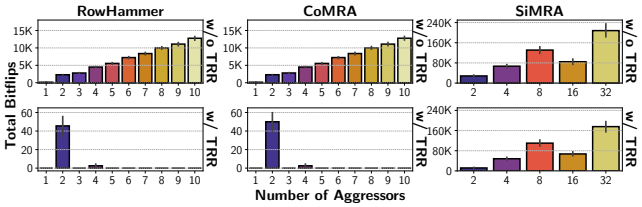

Figure 24: Number of bitflips in victim rows averaged across all tests when performing RowHammer, CoMRA, and SiMRA without TRR (top) and with TRR (bottom).

Observation 25. SiMRA and CoMRA bypass the TRR mechanism and induce more bitflips than RowHammer.

For example, SiMRA with 32-row activation and 2-sided CoMRA respectively induce 11340× and 1.10× more bitflips than 2-sided RowHammer on average across all 5 iterations.

Observation 26. TRR cannot effectively mitigate SiMRA-induced read disturbance bitflips.

We observe that SiMRA bitflips reduce only 15.62% on average with TRR, whereas RowHammer and CoMRA bitflips reduce greatly (e.g., 99.89% reduction on average for RowHammer). We hypothesize that this is due to two reasons. First, since SiMRA only issues two ACT commands back-to-back (i.e., ACT-PRE-ACT, see §2.3 and §5.1) to simultaneously activate up to 32 rows, the TRR mechanism likely can only mitigate the bitflips in the victims of the two aggressor row addresses issued with the two ACT commands. Second, since the HC_{first} of SiMRA is much lower (e.g., 26) than the maximum number of ACT commands we can issue in a periodic refresh window (i.e., 156), SiMRA induces read disturbance bitflips before the memory controller can issue a TRR-capable REF command.

Takeaway 9. SiMRA and CoMRA bypass in-DRAM TRR mechanism and are more effective than RowHammer to induce read disturbance bitflips.

8. Mitigating PuDHammer

We experimentally demonstrate the potential interaction between read disturbance and multiple-row activation-based PuD mechanisms. We perform multiple-row activation operations within the constraints of current DRAM chips. Today's DRAM chips are *not* designed to support these operations. Our results provide insights into read disturbance challenges that PuD-enabled systems might face and show that read disturbance vulnerability should be taken into account while designing PuD-enabled systems.[6]

Future work should investigate countermeasures in detail (based on specific PuD designs) and propose further countermeasures based on our findings. We leave a detailed investigation of such analyses and countermeasures for future work, but 1) sketch three types of countermeasures against PuD-based read disturbance bitflips and 2) adapt and evaluate the state-

[5] The memory controller issues a REF once every 7.8μs. This allows at most 156 ACT commands to a single DRAM bank in the tested module in a refresh window.

[6] Future solutions to such challenges should include designing DRAM chips to fundamentally 1) support CoMRA and SiMRA and 2) mitigate read disturbance. Our analysis aids the development of such designs by providing new experimental insights.

of-the-art RowHammer mitigation standardized by industry, called Per Row Activation Counting (PRAC) [159–161, 163], to account for PuDHammer.

8.1. Countermeasures Against PuDHammer

Separating PuD-enabled rows. Prior PuD architectures [29, 32, 80, 225] split a DRAM array into two parts, where a small set of rows are used for computation (compute region) and the remaining large fraction of rows for storing data (storage region). In such an architecture, CoMRA and SiMRA operations would happen repeatedly in the compute region, resulting in activation counts exceeding what we experimentally demonstrated to cause read disturbance bitflips (e.g., to perform an 8-bit multiplication operation, SIMDRAM [32] issues 663 CoMRA and SiMRA operations). Future PuD systems can be designed with two constraints: 1) SiMRA operations are allowed *only* on the compute region rows, and 2) at most one of the operands in CoMRA operations (source and destination rows) can be a row outside of the compute region.

These two constraints 1) limit SiMRA's and double-sided CoMRA's negative impact on read disturbance to the compute region and 2) limit the HC_{first} decrease on storage region rows to the reduction of single-sided CoMRA operations. To prevent read disturbance bitflips in the compute region, each row in the relatively small compute region (e.g., 3 to 32 rows [29, 32, 79, 80, 225] out of 1024 rows in a DRAM subarray [78, 79, 195, 216, 218, 226]) can be periodically refreshed after a number of SiMRA operations (e.g., 20). These refresh operations can be spread over time similarly to how periodic refreshes are performed (i.e., after each SiMRA operation, a portion of rows in the compute region are refreshed). To prevent read disturbance bitflips outside the compute region, existing RowHammer mitigation mechanisms can be simply configured for a reduced HC_{first} value as the reduction due to single-sided CoMRA is less than 2% (Fig. 7). Doing so is possible at low hardware complexity as this approach does *not* require sophisticated tracking mechanisms, however, it might cause performance and energy overheads.

Weighted contribution of different row activation types. Based on the HC_{first} reduction factors of CoMRA and SiMRA, each CoMRA or SiMRA operation can be accounted for an equivalent hammer count using double-sided hammering, e.g., assuming that CoMRA reduces HC_{first} by 20×, each CoMRA operation can increment row activation counters by 20. Prior works [153, 227] propose a similar approach to prevent RowPress bitflips by slightly modifying existing RowHammer mitigation mechanisms.

Clustered multiple-row activation. We demonstrate that double-sided SiMRA significantly reduces HC_{first} (§5). This operation is possible because the row decoder circuitry simultaneously activates multiple rows in a variety of locations within the subarray (as shown in prior work [79]), thereby causing some of the simultaneously activated rows to sandwich an unactivated row. Future PuD-enabled systems can mitigate double-sided SiMRA by employing row decoder circuits that cluster simultaneous row activation by guaranteeing that adjacent rows are activated, i.e., there are no sandwiched unactivated rows.

8.2. Adapting Existing RowHammer Mitigations

In this section, we adapt and evaluate the state-of-the-art industry solution to RowHammer (i.e., PRAC [159–161, 163]) to mitigate PuDHammer. We demonstrate that adapted PRAC incurs 48.26% average system performance overhead across all tested workloads.

PRAC Overview. PRAC [159–161, 163], as described in the JEDEC DDR5 standard updated in April 2024 [163], implements a counter for each row. Upon an `ACT`, PRAC increases the counter of the activated row and thus accurately measures the activation counts of all rows. When a row's activation count reaches a threshold (the read disturbance threshold (RDT) is the minimum number of activations needed to induce a read disturbance bitflip in a victim row), the DRAM chip asserts a back-off signal [159–161, 163], which forces the memory controller to issue a command called RFM [228]. The DRAM chip preventively refreshes potential victim rows upon receiving an RFM command.[7]

Key Challenge: Updating Multiple Counters in PRAC. A SiMRA operation activates multiple rows *simultaneously* and thus requires updating *multiple* PRAC row activation counters simultaneously to prevent SiMRA-induced read disturbance bitflips. However, since PRAC is designed to update *only* one counter upon an `ACT` command (i.e., the one corresponding to the activated row), we need to modify the counter organization and operation in PRAC to support updating multiple counters. We assume a PRAC implementation described in Panopticon [231] where the counters reside in different subarrays than the rows that store data.[8] We provide new methods to properly update multiple counters in PRAC: 1) an area-optimized solution and 2) a performance-optimized solution.

Area-Optimized Solution: PRAC-AO. In the area-optimized solution (PRAC-AO), we sequentially update the activation counters of each simultaneously activated row (i.e., update counters one by one). Doing so requires only one incrementer circuitry in the DRAM chip to update counters and a small subarray (or mat) where row activation counters are kept (separately from the simultaneously activated rows), as proposed by Panopticon. However, since we need to update multiple (e.g., up to 32) counters, the latency for updating all counters becomes significantly higher than the standard memory access latency (t_{RC}, typically $46\,\text{ns}$-$50\,\text{ns}$). For example, if the SiMRA operation activates 32 rows simultaneously, the mitigation mechanism needs to update 32 counters, which translates to a latency of $32 \times t_{RC}$ (approximately $1.5\,\mu\text{s}$-$1.6\,\mu\text{s}$).

PRAC-AO likely induces prohibitive performance overheads due to the significant counter update latency. If we employ this solution as the read disturbance mitigation mechanism, 1)

[7] We refer the reader to recent works [159–161, 163, 229, 230] for more detail on PRAC.

[8] If counters are placed in the same subarray as the simultaneously activated DRAM rows, the counters that are simultaneously activated lose their values as SiMRA is a destructive operation that overwrites all activated rows with the result of the analog majority operation. Therefore, PRAC becomes insecure.

SiMRA operation throughput likely reduces significantly, and 2) every SiMRA operation blocks the target DRAM bank for ≈1.5 µs during which the memory controller cannot serve any memory request targeting the same bank. Such performance degradation would defeat the purpose of using PuD operations. To avoid this, we propose and evaluate a performance-optimized solution (PRAC-PO) that does *not* suffer from the drawbacks of the area-optimized solution.

Performance-Optimized Solution: PRAC-PO. We *simultaneously* update the activation counters of all simultaneously activated rows (i.e., update all counters at once). By doing so, we can alleviate the significant counter update latency of the area-optimized solution (e.g., reducing $32\times t_{RC}$ to t_{RC}). Assuming we can activate up to N rows simultaneously, this solution requires 1) simultaneous access to N different counter values, and 2) N incrementer circuits to update all the counters simultaneously. We leave a detailed area overhead evaluation of this technique to future work.

Weighted Counting Optimization. Due to prohibitive performance overheads of PRAC-AO as discussed in the PRAC-PO section, we optimize and focus our analysis on PRAC-PO. PRAC-PO securely prevents all read disturbance bitflips when configured for an RDT of ≈20 to account for SiMRA-induced read disturbance failures. However, this is a conservative configuration of our performance-optimized solution: the configuration induces prohibitive system performance overheads (as shown in Fig. 25). This is because the solution does *not* take the heterogeneity in the read disturbance effects of RowHammer, CoMRA operations, and SiMRA operations. We find that the lowest HC_{first} values for RowHammer, CoMRA, and SiMRA are ≈4K, ≈400, and ≈20, respectively. Leveraging this heterogeneity, we propose an optimization to reduce the performance overheads of PRAC: weighted contribution of different row activation types. We assign each operation (i.e., RowHammer, SiMRA, and CoMRA) a weight equal to the lowest HC_{first} for RowHammer divided by that for the operation (e.g., 4K/20 = 200 for SiMRA and 4K/400 = 10 for CoMRA). Thus, we count each SiMRA operation as 200 hammers (e.g., by adding 200 to the counters of simultaneously activated rows) and each CoMRA operation as 10 hammers to each row that participates in the SiMRA or CoMRA operation where 1 hammer is an activation of a DRAM row.

Evaluation Methodology. To evaluate performance, we conduct cycle-level simulations using Ramulator 2.0 [232, 233] (new version of Ramulator [234, 235]) with a realistic baseline system configuration.[9] We extend Ramulator 2.0 to support SiMRA and CoMRA operations and update multiple (up to 32) counters simultaneously for PRAC-PO. We execute 60 five-core multiprogrammed workload mixes. Each mix is composed of four workloads from five major benchmark suites [237–241] and one synthetic workload that periodically performs back-to-back one SiMRA with 32-row activation and one CoMRA operation every N ns (where N ranges from 125ns to 16μs).[10] We evaluate system performance using the weighted speedup metric [242, 243].

We analyze two different PRAC-PO implementations and evaluate their overheads on the system performance: 1) *PRAC-PO-Naive*: a naive PRAC-PO implementation without weighted counting optimization where the RowHammer threshold is reduced to lowest observed HC_{first} for SiMRA (i.e., 20) and 2) *PRAC-PO-Weighted Counting (PRAC-PO-WC)*: PRAC-PO with weighted counting optimization.

Results. Fig. 25 presents the performance overheads of the evaluated two PRAC-PO implementations, i.e., PRAC-PO-Naive and PRAC-PO-WC, across 60 five-core multiprogrammed workload mixes. The x-axis shows the synthetic PuD workload's period of performing one SiMRA and one CoMRA operation (lower period indicates higher PuD operation intensity), and the y-axis shows normalized system performance (higher is better) in terms of weighted speedup normalized to a baseline with *no* read disturbance mitigation.

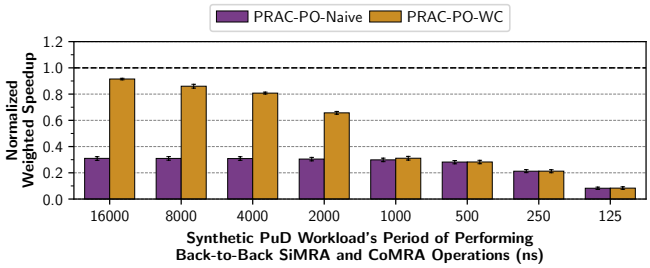

Figure 25: Performance impact of evaluated PRAC-PO implementations on 60 five-core multiprogrammed workloads.

We make two major observations from Fig. 25. First, even with our proposed optimizations, PRAC-PO-WC incurs non-negligible performance overhead to mitigate PuDHammer-based read disturbance bitflips. We observe that PRAC-PO-WC incurs an average (maximum) system performance overhead of 48.26% (98.83%) across all tested PuD operation intensity rates and workloads. Second, across all evaluated PuD operation intensity rates, PRAC-PO-WC outperforms PRAC-PO-Naive. For example, at a period of 4μs, PRAC-PO-WC induces 19.26% average system performance overhead, whereas PRAC-PO-Naive induces 69.15%.

We conclude that even with our proposed optimizations, PRAC-PO-WC incurs significant performance overheads at high PuD operation intensity and non-negligible performance overheads at low PuD operation intensity to mitigate PuDHammer. We expect future work to introduce new efficient and effective mitigation mechanisms, as it has been happening analogously for RowHammer and RowPress.

9. Related Work

We present the first experimental characterization of the read disturbance vulnerability caused by both consecutive multiple-row activation (CoMRA) and simultaneous multiple-row activation (SiMRA) in real DDR4 DRAM chips.

[9] 4.2GHz 5-core system, dual-rank DDR5 DRAM, FR-FCFS+Cap of 4 [236]. We simulate each workload until every core execute 100M instructions.

[10] We vary the period to show how performance changes with the workload's PuD operation intensity.

Read Disturbance Characterization. Many works [93, 137–141, 144–146, 153, 154, 211, 214, 215, 218, 219, 244–247] experimentally demonstrate (using real DDR3, DDR4, LPDDR4, and HBM2 DRAM chips) how a DRAM chip's read disturbance vulnerability varies with 1) DRAM refresh rate [93, 119, 125], 2) the physical distance between aggressor and victim rows [93, 144, 245], 3) DRAM generation and technology node [93, 125, 144, 145], 4) temperature [138, 145], 5) the time the aggressor row stays active [138, 145, 153, 216, 218, 246, 248], 6) physical location of the victim DRAM cell [145, 211, 216, 218], 7) wordline voltage [146], and 8) supply voltage [247]. Our results are significantly different from what is already well-established in prior read disturbance characterization works. We study the read disturbance effects of *processing-using-DRAM* operations' row activation patterns, which are fundamentally different from those studied by prior read disturbance characterization works: all prior read disturbance characterization works [93, 137–141, 144–146, 153, 154, 211, 214, 215, 218, 219, 244–247] activate *at most one* aggressor row *at a given time*. In contrast, this paper investigates the read disturbance effect of activating **multiple** aggressor rows *simultaneously* (i.e., SiMRA) or in *quick succession* (i.e., CoMRA). Our results demonstrate that the read disturbance effects of these new patterns (i.e., SiMRA and CoMRA operations) are fundamentally more damaging. In this work, we show that they can induce bitflips with *only* 26 operations (Fig. 13), which take 1.48μs, whereas the best known RowHammer pattern induces bitflips with 4123 aggressor row activations (Fig. 13), which takes 210.27μs and the best tested RowPress pattern (in terms of minimum HC_{first}) induces bitflips with 37 aggressor row activations at t_{AggOn}=70.2μs (Fig. 13), which takes 2597.4μs. *No* prior read disturbance characterization study evaluates the read disturbance effect of SiMRA and CoMRA access patterns.

Multiple-Row Activation-based PuD Operations in COTS Chips. Several prior works demonstrate bulk bitwise [29, 30, 32, 80, 194] and bulkdata copy operations [40] in COTS DRAM chips using multiple-row activation [33, 73–78, 195]. ComputeDRAM [73] activates three rows simultaneously to perform three-input majority and two-input AND and OR operations, and 2) demonstrates copying one row's content to another row in DDR3 chips. FracDRAM [74] shows that a DRAM cell in DDR3 chips can store fractional values. QUAC-TRNG [76] simultaneously activates four rows to generate true random numbers in DDR4 chips and a recent work [33, 249] experimentally studies the simultaneous activation of 2, 8, 16, and 32 rows in a subarray in COTS DDR4 DRAM chips. HiRA [195] demonstrates that DDR4 chips can activate two rows in quick succession in electrically isolated subarrays. DRAM Bender [75] demonstrates two-input AND and OR operations in DDR4 chips. PiDRAM [77] provides an FPGA-based framework that enables real system studies of PuD techniques (e.g., RowClone). We do not use PiDRAM because PiDRAM is not designed to test DRAM chips but instead to evaluate PuD applications running end-to-end in a computing system. PiDRAM also performs SiMRA and CoMRA operations inside the DRAM chip, and thus it also suffers from the read disturbance effect that we investigate. Prior work [78] demonstrates NOT and up to 16-input AND, NAND, OR, and NOR operations by simultaneously activating up to 48 rows in neighboring subarrays. A recent work [79] demonstrates up to 9-input majority operations and copying one row's content to up to 31 other rows concurrently by simultaneously activating up to 32 rows. None of these works investigates the read disturbance effects of such multiple-row activation operations.

10. Conclusion

We presented our extensive characterization study on the interaction between read disturbance and multiple-row activation-based Processing-using-DRAM operations in 316 COTS DDR4 DRAM chips from four major manufacturers. Our study leads to 26 new empirical observations and shares 9 key takeaway lessons, which demonstrate that multiple-row activation significantly exacerbates the DRAM read disturbance vulnerability either by itself or when combined with RowHammer, and this vulnerability gets worse under various operating conditions and parameters (e.g., data pattern). We discuss four countermeasures against exploiting the read disturbance effect of PuD for future PuD-enabled systems. We hope and expect that our detailed characterization results motivate and guide both 1) system-level and architectural solutions to enable read-disturbance-resilient PuD systems and 2) silicon-level works in understanding the underlying physical phenomena that explain PuDHammer's characteristics.

Acknowledgments

We thank the anonymous reviewers of ISCA 2025 for their feedback. We thank the SAFARI Research Group members for providing a stimulating intellectual and scientific environment. We acknowledge the generous gifts from our industrial partners, including Google, Huawei, Intel, and Microsoft. This work, along with our broader work in Processing-in-Memory and memory systems, is supported in part by the Semiconductor Research Corporation (SRC), the ETH Future Computing Laboratory (EFCL), AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Hong Kong SAR, European Union's Horizon programme for research and innovation [101047160 - BioPIM], a Google Security and Privacy Research Award, and the Microsoft Swiss Joint Research Center.

References

[1] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," in *Microprocessors and Microsystems*, 2019.
[2] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A Modern Primer on Processing in Memory," in *Emerging Computing: From Devices to Systems — Looking Beyond Moore and Von Neumann*. Springer, 2021. [Online]. Available: https://arxiv.org/abs/2012.03112
[3] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[4] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[5] J. Dean and L. A. Barroso, "The Tail at Scale," *CACM*, 2013.
[6] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.
[7] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware," in *ASPLOS*, 2012.

[8] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A Big Data Benchmark Suite from Internet Services," in *HPCA*, 2014.

[9] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling Practical Processing in and Near Memory For Data-Intensive Computing," in *DAC*, 2019.

[10] O. Mutlu, "Intelligent Architectures for Intelligent Machines," in *VLSI-DAT*, 2020.

[11] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, 2019.

[12] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *IEEE Access*, 2021.

[13] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.

[14] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," in *PACT*, 2021.

[15] S. Wang and E. Ipek, "Reducing Data Movement Energy via Online Data Clustering and Encoding," in *MICRO*, 2016.

[16] D. Pandiyan and C.-J. Wu, "Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms," in *IISWC*, 2014.

[17] S. Koppula, L. Orosa, A. G. Yağlıkçı, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, "EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM," in *MICRO*, 2019.

[18] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.

[19] S. A. McKee *et al.*, "Reflections on the Memory Wall." *CF*, 2004.

[20] M. V. Wilkes, "The Memory Gap and the Future of High Performance Memories," *CAN*, 2001.

[21] Y. Kim, V. Seshadri, D. Lee, J. Liu, O. Mutlu, Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[22] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *CAN*, 1995.

[23] S. Ghose, T. Li, N. Hajinazar, D. S. Cali, and O. Mutlu, "Demystifying Complex Workload–DRAM Interactions: An Experimental Study," in *SIGMETRICS*, 2020.

[24] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[25] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.

[26] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.

[27] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi *et al.*, "FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching," in *MICRO*, 2020.

[28] R. Sites, "It's the Memory, Stupid!" *MPR*, 1996.

[29] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[30] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch†, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," in *CAL*, 2015.

[31] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," *arXiv:1905.09822*, 2019.

[32] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM," in *ASPLOS*, 2021.

[33] O. Mutlu, A. Olgun, G. F. Oliveira, and I. E. Yuksel, "Memory-Centric Computing: Recent Advances in Processing-in-DRAM," in *IEDM*, 2024.

[34] O. Mutlu, A. Olgun, and I. E. Yuksel, "Memory-Centric Computing: Solving Computing's Memory Problem," in *IMW*, 2025.

[35] C.-Y. Chan and Y. E. Ioannidis, "Bitmap Index Design and Evaluation," in *SIGMOD*, 1998.

[36] E. O'Neil, P. O'Neil, and K. Wu, "Bitmap Index Design Choices and Their Performance Implications," in *IDEAS*, 2007.

[37] Y. Li and J. M. Patel, "WideTable: An Accelerator for Analytical Data Processing," *VLDB*, 2014.

[38] ——, "BitWeaving: Fast Scans for Main Memory Data Processing," in *SIGMOD*, 2013.

[39] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He, "BitFunnel: Revisiting Signatures for Search," in *SIGIR*, 2017.

[40] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[41] K. Wu, "FastBit: An Efficient Indexing Technology for Accelerating Data-Intensive Science," in *Journal of Physics: Conference Series*, 2005.

[42] M.-C. Wu and A. P. Buchmann, "Encoded Bitmap Indexing for Data Warehouses," in *ICDE*, 1998.

[43] Redis, "Redis bitmaps," https://redis.io/docs/data-types/bitmaps/.

[44] B. Perach, R. Ronen, B. Kimelfeld, and S. Kvatinsky, "Understanding Bulk-Bitwise Processing In-Memory Through Database Analytics," *ETC*, 2023.

[45] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, and S. Xu, "Bluedbm: An Appliance for Big Data Analytics," *ISCA*, 2015.

[46] M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Catalina: In-Storage Processing Acceleration for Scalable Big Data Analytics," in *PDP*, 2019.

[47] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD," *CAL*, 2020.

[48] M. Besta, R. Kanakagiri, G. Kwasniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan *et al.*, "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *MICRO*, 2021.

[49] S. Beamer, K. Asanovic, and D. Patterson, "Direction-Optimizing Breadth-First Search," in *SC*, 2012.

[50] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.

[51] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "Parabit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs," in *MICRO*, 2021.

[52] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "Gatekeeper: A New Hardware Architecture For Accelerating Pre-Alignment In DNA Short Read Mapping," in *Bioinformatics*, 2017.

[53] J. Loving, Y. Hernandez, and G. Benson, "BitPAl: A Bit-Parallel, General Integer-Scoring Sequence Alignment Algorithm," *Bioinformatics*, 2014.

[54] H. Xin, J. Greth, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu, "Shifted Hamming Distance: A Fast and Accurate SIMD-Friendly Filter to Accelerate Alignment Verification in Read Mapping," *Bioinformatics*, 2015.

[55] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *MICRO*, 2020.

[56] J. S. Kim, D. Senol, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast Seed Filtering in Read Mapping using Emerging Memory Technologies," in *APBC*, 2017.

[57] G. Myers, "A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming," *JACM*, 1999.

[58] J. Han, C.-S. Park, D.-H. Ryu, and E.-S. Kim, "Optical Image Encryption Based on XOR Operations," *Optical Engineering*, 1999.

[59] P. Tuyls, H. D. Hollmann, J. V. Lint, and L. Tolhuizen, "XOR-based Visual Cryptography Schemes," *Des. Codes, Cryptogr.*, 2005.

[60] P. Kanerva, "Sparse Distributed Memory and Related Models," Tech. Rep., 1992.

[61] ——, "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors," *Cognitive Computation*, 2009.

[62] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory Hyperdimensional Computing," *Nature Electronics*, 2020.

[63] Y. He, H. Mao, C. Giannoula, M. Sadrosadati, J. Gómez-Luna, H. Li, X. Li, Y. Wang, and O. Mutlu, "PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System," *ASPLOS*, 2025.

[64] Y. Gu, A. Khadem, S. Umesh, N. Liang, X. Servot, O. Mutlu, R. Iyer, and R. Das, "PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference," *ASPLOS*, 2025.

[65] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A Memory-Based Acceleration via Software-Hardware Co-Design for Transformer," in *HPCA*, 2022.

[66] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference," in *ASPLOS*, 2024.

[67] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin *et al.*, "IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System," in *ASPLOS*, 2024.

[68] S. Yun, K. Kyung, J. Cho, J. Choi, J. Kim, B. Kim, S. Lee, K. Sohn, and J. H. Ahn, "Duplex: A Device for Large Language Models with Mixture of Experts, Grouped Query Attention, and Continuous Batching," in *MICRO*, 2024.

[69] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim Heterogeneous Acceleration for Batched LLM Inferencing," in *ASPLOS*, 2024.

[70] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *NIPS*, 2020.

[71] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*, 2019.

[72] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *NIPS*, 2014.

[73] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.

[74] ——, "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in *MICRO*, 2022.

[75] A. Olgun, H. Hassan, A. G. Yağlıkçı, Y. C. Tuğrul, L. Orosa, H. Luo, M. Patel, O. Ergin, and O. Mutlu, "DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips," *TCAD*, 2023.

[76] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, N. Bostancı, N. Vijaykumar, O. Ergin, and O. Mutlu, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips," in *ISCA*, 2021.

[77] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, and O. Mutlu,

"PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM," *TACO*, 2022.
[78] I. E. Yuksel, Y. C. Tugrul, A. Olgun, F. N. Bostanci, A. G. Yaglikci, G. F. de Oliveira, H. Luo, J. G. Luna, M. Sadrosadati, and O. Mutlu, "Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis," in *HPCA*, 2024.
[79] I. E. Yuksel, Y. C. Tugrul, F. N. Bostanci, G. F. de Oliveira, A. G. Yaglikci, A. Olgun, M. Soysal, H. Luo, J. G. Luna, M. Sadrosadati, and O. Mutlu, "Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis," in *DSN*, 2024.
[80] G. F. Oliveira, A. Olgun, A. G. G. Yaglikçi, N. Bostanci, J. Gómez-Luna, S. Ghose, and O. Mutlu, "MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Processing," HPCA, 2024.
[81] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," arXiv, 2016.
[82] V. Seshadri and O. Mutlu, "The Processing Using Memory Paradigm: In-DRAM Bulk Copy, Initialization, Bitwise AND and OR," arXiv:1610.09603, 2016.
[83] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: Accelerating Data Movement and Initialization Using DRAM," arXiv, 2018.
[84] Z. Jahshan and L. Yavits, "MajorK: Majority Based Kmer Matching in Commodity DRAM," *CAL*, 2024.
[85] A. P. Fournaris, L. Pocero Fraile, and O. Koufopavlou, "Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks," *Electronics*, 2017.
[86] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking Deterministic Signature Schemes using Fault Attacks," in *EuroS&P*, 2018.
[87] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks Over the Network and Defenses," in *USENIX ATC*, 2018.
[88] S. Carre, M. Desjardins, A. Facon, and S. Guilley, "OpenSSL Bellcore's Protection Helps Fault Attack," in *DSD*, 2018.
[89] A. Barenghi, L. Breveglieri, N. Izzo, and G. Pelosi, "Software-Only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks," in *IVSW*, 2018.
[90] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ASHES*, 2018.
[91] S. Bhattacharya and D. Mukhopadhyay, "Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug," in *Fault Tolerant Architectures for Cryptography and Hardware Security*, 2018.
[92] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.
[93] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[94] SAFARI Research Group, "RowHammer — GitHub Repository," https://github.com/CMU-SAFARI/rowhammer, 2014.
[95] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," *Black Hat*, 2015.
[96] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.
[97] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript," in *DIMVA*, 2016.
[98] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.
[99] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.
[100] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.
[101] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector," in *S&P*, 2016.
[102] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis," in *CHES*, 2016.
[103] W. Burleson, O. Mutlu, and M. Tiwari, "Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer," in *DAC*, 2016.
[104] R. Qiao and M. Seaborn, "A New Approach for RowHammer Attacks," in *HOST*, 2016.
[105] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory," in *USENIX Security*, 2017.
[106] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SOSP*, 2017.
[107] M. T. Aga, Z. B. Aweke, and T. Austin, "When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks," in *HOST*, 2017.
[108] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[109] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *RAID*, 2018.
[110] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *S&P*, 2018.
[111] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing Rowhammer Faults Through Network Requests," arXiv:1805.04956 [cs.CR], 2018.
[112] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.
[113] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *S&P*, 2018.
[114] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.
[115] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks," in *ASIACCS*, 2019.
[116] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," *TCAD*, 2019.
[117] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks," in *USENIX Security*, 2019.
[118] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *S&P*, 2020.
[119] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.
[120] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *S&P*, 2020.
[121] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "JackHammer: Efficient Rowhammer on Heterogeneous FPGA–CPU Platforms," arXiv:1912.11523 [cs.CR], 2020.
[122] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses," in *MICRO*, 2020.
[123] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the Intelligence of Deep Neural Networks Through Targeted Chain of Bit Flips," in *USENIX Security*, 2020.
[124] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript," in *USENIX Security*, 2021.
[125] H. Hassan, Y. C. Tugrul, J. S. Kim, V. v. d. Veen, K. Razavi, and O. Mutlu, "Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021.
[126] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *S&P*, 2022.
[127] M. C. Tol, S. Islam, B. Sunar, and Z. Zhang, "Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer," arXiv:2110.07683, 2022.
[128] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering From the Next Row Over," in *USENIX Security*, 2022.
[129] L. Orosa, U. Rührmair, A. G. Yaglikci, H. Luo, A. Olgun, P. Jattke, M. Patel, J. Kim, K. Razavi, and O. Mutlu, "SpyHammer: Using RowHammer to Remotely Spy on Temperature," *IEEE Access*, 2022.
[130] Z. Zhang, W. He, Y. Cheng, W. Wang, Y. Gao, D. Liu, K. Li, S. Nepal, A. Fu, and Y. Zou, "Implicit Hammer: Cross-Privilege-Boundary Rowhammer through Implicit Accesses," *IEEE TDSC*, 2022.
[131] L. Liu, Y. Guo, Y. Cheng, Y. Zhang, and J. Yang, "Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack," *IEEE TC*, 2022.
[132] Y. Cohen, K. S. Tharayil, A. Haenel, D. Genkin, A. D. Keromytis, Y. Oren, and Y. Yarom, "HammerScope: Observing DRAM Power Consumption Using Rowhammer," in *CCS*, 2022.
[133] M. Zheng, Q. Lou, and L. Jiang, "TrojViT: Trojan Insertion in Vision Transformers," arXiv:2208.13049, 2022.
[134] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer," *CCS*, 2022.
[135] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *S&P*, 2022.
[136] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories," in *S&P*, 2022.
[137] K. Park, D. Yun, and S. Baeg, "Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components," *Microelectronics Reliability*, 2016.
[138] K. Park, C. Lim, D. Yun, and S. Baeg, "Experiments and Root Cause Analysis for Active-Precharge Hammering Fault in DDR3 SDRAM under 3xnm Technology," *Microelectronics Reliability*, 2016.
[139] C. Lim, K. Park, and S. Baeg, "Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM," *TNS*, 2017.
[140] S.-W. Ryu, K. Min, J. Shin, H. Kwon, D. Nam, T. Oh, T.-S. Jang, M. Yoo, Y. Kim, and S. Hong, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing," in *IEDM*, 2017.
[141] D. Yun, M. Park, C. Lim, and S. Baeg, "Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs," in *IRPS*, 2018.
[142] T. Yang and X.-W. Lin, "Trap-Assisted DRAM Row Hammer Effect," *EDL*, 2019.
[143] A. J. Walker, S. Lee, and D. Beery, "On DRAM RowHammer and the Physics on Insecurity," *IEEE TED*, 2021.
[144] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques," in *ISCA*, 2020.
[145] L. Orosa, A. G. Yağlıkçı, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and

O. Mutlu, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," in *MICRO*, 2021.
[146] A. G. Yağlıkcı, H. Luo, G. F. De Oliviera, A. Olgun, M. Patel, J. Park, H. Hassan, J. S. Kim, L. Orosa, and O. Mutlu, "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices," in *DSN*, 2022.
[147] M. N. I. Khan and S. Ghosh, "Analysis of Row Hammer Attack on STTRAM," in *ICCD*, 2018.
[148] S. Agarwal, H. Dixit, D. Datta, M. Tran, D. Houssameddine, D. Shum, and F. Benistant, "Rowhammer for Spin Torque based Memory: Problem or not?" in *INTERMAG*, 2018.
[149] H. Li, H.-Y. Chen, Z. Chen, B. Chen, R. Liu, G. Qiu, P. Huang, F. Zhang, Z. Jiang, B. Gao, L. Liu, X. Liu, S. Yu, H.-S. P. Wong, and J. Kang, "Write Disturb Analyses on Half-Selected Cells of Cross-Point RRAM Arrays," in *IRPS*, 2014.
[150] K. Ni, X. Li, J. A. Smith, M. Jerry, and S. Datta, "Write Disturb in Ferroelectric FETs and Its Implication for 1T-FeFET AND Memory Arrays," *IEEE EDL*, 2018.
[151] P. R. Genssler, V. M. van Santen, J. Henkel, and H. Amrouch, "On the Reliability of FeFET On-Chip Memory," *TC*, 2022.
[152] O. Mutlu, A. Olgun, and A. G. Yaglikci, "Fundamentally Understanding and Solving RowHammer," in *ASP-DAC*, 2023.
[153] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *ISCA*, 2023.
[154] H. Luo, I. E. Yüksel, A. Olgun, A. G. Yağlıkçı, M. Sadrosadati, and O. Mutlu, "An Experimental Characterization of Combined RowHammer and RowPress Read Disturbance in Modern DRAM Chips," in *DSN Disrupt*, 2024.
[155] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Adv. Comput.*, 2017.
[156] Micron, "DDR4 SDRAM Datasheet," in *Micron*, 2016, p. 380.
[157] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, S. Nepal, Y. Gao, K. Li, Z. Wang, and C. Wu, "SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh," in *USENIX ATC*, 2022.
[158] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "ProTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *S&P*, 2022.
[159] S. Saroiu, "DDR5 Spec Update Has All It Needs to End Rowhammer: Will It?" https://stefan.t8k2.com/rh/PRAC/index.html.
[160] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, N. Bostancı, I. E. Yuksel, H. Luo, O. Ergin, and O. Mutlu, "Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance," in *HPCA*, 2025.
[161] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, O. Ergin, and O. Mutlu, "Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance," *DRAMSec*, 2024.
[162] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang *et al.*, "A 1.1 V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement," in *ISSCC*, 2023.
[163] JEDEC, *JESD79-5c: DDR5 SDRAM Standard*, 2024.
[164] ——, *JESD209-5A: LPDDR5 SDRAM Standard*, 2020.
[165] ——, *JESD209-4B: Low Power Double Data Rate 4 (LPDDR4) Standard*, 2017.
[166] ——, *JESD79F: Double Data Rate (DDR) SDRAM Standard*, 2008.
[167] JEDEC, *JESD79-4C: DDR4 SDRAM Standard*, 2020.
[168] J. S. S. T. Association *et al.*, "DDR3 SDRAM Standard," *JEDEC Standard*, no. 79-3F, p. 226, 2012.
[169] JEDEC, *JESD79-5C: DDR5 SDRAM Standard*, 2024.
[170] ——, *JESD235D: High Bandwidth Memory DRAM (HBM1, HBM2)*, 2021.
[171] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[172] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[173] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[174] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[175] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[176] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[177] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, O. Mutlu, J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," in *ISCA*, 2013.
[178] Micron Technology, "SDRAM, 4Gb: x4, x8, x16 DDR4 SDRAM Features," 2014.
[179] O. Mutlu, "Retrospective: Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors," *arXiv*, 2023.
[180] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[181] X. Xin, Y. Zhang, and J. Yang, "ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM," in *HPCA*, 2020.
[182] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.
[183] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference," in *DAC*, 2018.
[184] S. Angizi and D. Fan, "GraphiDe: A Graph Processing Accelerator Leveraging In-DRAM-Computing," in *GLSVLSI*, 2019.
[185] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in *MICRO*, 2018.
[186] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori, and O. Mutlu, "pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation," in *MICRO*, 2022.
[187] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "LAcc: Exploiting Lookup Table-Based Fast and Accurate Vector Multiplication in DRAM-Based CNN Accelerator," in *DAC*, 2019.
[188] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. Prajapati, M. A. Indovina, S. M. P. Dinakarrao, and A. Ganguly, "Look-Up-Table Based Processing-in-Memory Architecture with Programmable Precision-Scaling for Deep Learning Applications," *TPDS*, 2021.
[189] P. R. Sutradhar, M. Connolly, S. Bavikadi, S. M. P. Dinakarrao, M. A. Indovina, and A. Ganguly, "pPIM: A Programmable Processor-in-Memory Architecture with Precision-Scaling For Deep Learning," in *CAL*, 2020.
[190] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in *MICRO*, 2018.
[191] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.
[192] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.
[193] G. F. Oliveira, J. Gómez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, "Accelerating Neural Network Inference with Processing-in-DRAM: From the Edge to the Cloud," *IEEE Micro*, 2022.
[194] G. F. Oliveira, M. Kabra, Y. Guo, K. Chen, A. G. Yağlıkçı, M. Soysal, M. Sadrosadati, J. O. Bueno, S. Ghose, J. Gómez-Luna *et al.*, "Proteus: Achieving High-Performance Processing-Using-DRAM via Dynamic Precision Bit-Serial Arithmetic," *ICS*, 2025.
[195] A. G. Yağlikci, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips," in *MICRO*, 2022.
[196] SAFARI Research Group, "DRAM Bender — GitHub Repository," https://github.com/CMU-SAFARI/DRAM-Bender, 2022.
[197] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[198] SAFARI Research Group, "SoftMC — GitHub Repository," https://github.com/CMU-SAFARI/softmc, 2017.
[199] Xilinx Inc., "Xilinx Alveo U200 FPGA Board," https://www.xilinx.com/products/boards-and-kits/alveo/u200.html.
[200] Maxwell, "FT20X User Manual," https://www.maxwell-fa.com/upload/files/base/8/m/311.pdf.
[201] M. Patel, J. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics," in *MICRO*, 2020.
[202] M. Patel, G. F. de Oliveira Jr., and O. Mutlu, "HARP: Practically and Effectively Identifying Uncorrectable Errors in Main Memory Chips That Use On-Die ECC," in *MICRO*, 2021.
[203] R. T. Smith, J. D. Chlipala, J. F. Bindels, R. G. Nelson, F. H. Fischer, and T. F. Mantz, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," *JSSC*, 1981.
[204] M. Horiguchi, "Redundancy Techniques for High-Density DRAMs," in *ISIS*, 1997.
[205] B. Keeth and R. Baker, *DRAM Circuit Design: A Tutorial*. John Wiley & Sons, 2001.
[206] K. Itoh, *VLSI Memory Chip Design*. Springer, 2001.
[207] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses," in *MICRO*, 2015.
[208] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[209] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[210] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[211] A. G. Yağlıkçı, G. F. Oliveira, Y. C. Tuğrul, I. E. Yuksel, A. Olgun, H. Luo, and O. Mutlu, "Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions," in *HPCA*, 2024.
[212] A. van de Goor and I. Schanstra, "Address and Data Scrambling: Causes and Impact on Memory Tests," in *DELTA*, 2002.
[213] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[214] A. Olgun, F. N. Bostanci, I. E. Yuksel, O. Canpolat, H. Luo, G. F. Oliveira, A. G.

Yaglikci, M. Patel, and O. Mutlu, "Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance," in *HPCA*, 2025.
[215] Y. C. Tugrul, A. G. Yaglikci, I. E. Yuksel, A. Olgun, O. Canpolat, N. Bostanci, M. Sadrosadati, O. Ergin, and O. Mutlu, "Understanding RowHammer Under Reduced Refresh Latency: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions," in *HPCA*, 2025.
[216] A. Olgun, M. Osseiran, A. G. Yaglikci, Y. C. Tugrul, H. Luo, S. Rhyner, B. Salami, J. G. Luna, and O. Mutlu, "Read Disturbance in High Bandwidth Memory: A Detailed Experimental Study on HBM2 DRAM Chips," in *DSN*, 2024.
[217] L. Zhou, J. Li, Z. Qiao, P. Ren, Z. Sun, J. Wang, B. Wu, Z. Ji, R. Wang, K. Cao, and R. Huang, "Double-sided Row Hammer Effect in Sub-20 nm DRAM: Physical Mechanism, Key Features and Mitigation," in *IRPS*, 2023.
[218] A. Olgun, M. Osseiran, A. G. Yaglikci, Y. C. Tugrul, H. Luo, S. Rhyner, B. Salami, J. Gomez Luna, and O. Mutlu, "An Experimental Analysis of RowHammer in HBM2 DRAM Chips," in *DSN Disrupt*, 2023.
[219] H. Luo, İ. E. Yüksel, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Revisiting DRAM Read Disturbance: Identifying Inconsistencies Between Experimental Characterization and Device-Level Studies," in *VTS*, 2025.
[220] J. Li, L. Zhou, S. Ye, Z. Qiao, and Z. Ji, "Understanding the Competitive Interaction in Leakage Mechanisms for Effective Row Hammer Mitigation in Sub-20 nm DRAM," *IEEE Electron Device Letters*, 2024.
[221] L. Zhou, J. Li, P. Ren, S. Ye, D. Wang, Z. Qiao, and Z. Ji, "Understanding the Physical Mechanism of RowPress at the Device-Level in Sub-20 nm DRAM," in *IRPS*, 2024.
[222] L. Zhou, S. Ye, R. Wang, and Z. Ji, "Unveiling RowPress in Sub-20 nm DRAM Through Comparative Analysis With Row Hammer: From Leakage Mechanisms to Key Features," in *IEEE TED*, 2024.
[223] S. Hynix, "DDR4 SDRAM Unbuffered DIMM Based on 8Gb A-die HMA81GU7AFR8N-UH Module – Datasheet ," https://gzhls.at/blob/ldb/6/a/7/1/0c4ba46b0049c17756d92c180858965180a7.pdf, 2016.
[224] "U-TRR," https://github.com/CMU-SAFARI/u-trr, 2021.
[225] M. F. Ali, A. Jaiswal, and K. Roy, "In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology," in *TCAS I*, 2019.
[226] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," *ISCA*, 2017.
[227] M. Qureshi, A. Saxena, and A. Jaleel, "ImPress: Securing DRAM Against Data-Disturbance Errors via Implicit Row-Press Mitigation," *MICRO*, 2024.
[228] JEDEC, *JESD79-5: DDR5 SDRAM Standard*, 2020.
[229] O. Canpolat, A. G. Yağlıkçı, A. Olgun, İ. E. Yüksel, Y. C. Tuğrul, K. Kanellopoulos, O. Ergin, and O. Mutlu, "BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads," *MICRO*, 2024.
[230] H. Hassan, A. Olgun, A. G. Yağlıkçı, H. Luo, and O. Mutlu, "Self-Managing DRAM: A Low-Cost Framework for Enabling Autonomous and Efficient DRAM Maintenance Operations," in *MICRO*, 2024.
[231] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," in *DRAMSec*, 2021.
[232] S. R. Group, "Ramulator V2.0," https://github.com/CMU-SAFARI/ramulator2.
[233] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, , and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," 2023.
[234] SAFARI Research Group, "Ramulator — GitHub Repository," https://github.com/CMU-SAFARI/ramulator.
[235] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2016.
[236] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[237] Standard Performance Evaluation Corp., "SPEC CPU 2006," http://www.spec.org/cpu2006/.
[238] ——, "SPEC CPU2017 Benchmarks," http://www.spec.org/cpu2017/.
[239] *Transaction Processing Performance Council*, TPC Benchmarks, http://www.tpc.org/information/benchmarks.asp.
[240] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, "MediaBench II Video: Expediting the next Generation of Video Systems Research," *Microprocess. Microsyst.*, 2009.
[241] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *SoCC*, 2010.
[242] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.
[243] A. Snavely and D. M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Mutlithreading Processor," in *ASPLOS*, 2000.
[244] C. Lim, K. Park, G. Bak, D. Yun, M. Park, S. Baeg, S.-J. Wen, and R. Wong, "Study of Proton Radiation Effect to Row Hammer Fault in DDR4 SDRAMs," *Microelectronics Reliability*, 2018.
[245] Z. Lang, P. Jattke, M. Marazzi, and K. Razavi, "Blaster: Characterizing the blast radius of rowhammer," in *DRAMSec*, 2023.
[246] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "Dramscope: Uncovering DRAM Microarchitecture and Characteristics by Issuing Memory Commands," *ISCA*, 2024.
[247] W. He, Z. Zhang, Y. Cheng, W. Wang, W. Song, Y. Gao, Q. Zhang, K. Li, D. Liu, and S. Nepal, "WhistleBlower: A System-level Empirical Study on RowHammer," *TC*, 2023.
[248] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "X-ray: Discovering DRAM Internal Structure and Error Characteristics by Issuing Memory Commands," *IEEE CAL*, 2023.
[249] I. E. Yuksel, A. Olgun, F. N. Bostanci, O. Canpolat, G. F. Oliveira, M. Maakenkova, M. Sadrosadati, A. G. Yaglikci, and O. Mutlu, "Experimental Analysis of True Random Number Generation Using Simultaneous Multiple-Row Activation in Real DRAM Chips," arXiv, 2025.
[250] TimeTec, "TIMETEC PREMIUM DDR4 SODIMM Laptop Memory," https://timeteci nc.com/products/timetec-premium-ddr4-sodimm-laptop-memory.
[251] SK Hynix, "4Gb DDR4 SDRAM H5AN4G8NAFR-xxC H5AN4G8NAFR-xxI H5AN4G6NAFR-xxC H5AN4G6NAFR-xxI," https://www.datasheets360.com/pdf/6330941288817950380 4, 2017.
[252] ——, "DDR4 SDRAM Unbuffered DIMM Based on 8Gb A-die," https://gzhls.at/blob/ldb/6/a/7/1/0c4ba46b0049c17756d92c180858965180a7.pdf, 2016.
[253] ——, "8Gb DDR4 SDRAM H5AN8G4NAFR-xxCH5AN8G8NAFR-xxCH5AN8G6NAFR-xxC," https://www.alldatasheet.com/datasheet-pdf/view/1424933/HYNIX/H5AN8G8NAFR-UHC.html, 2017.
[254] Kingston, "KSM26ES8/16HC 16GB 1Rx8 2G x 72-Bit PC4-2666 CL19 288-Pin DIMM," https://www.kingston.com/datasheets/KSM26ES8_16HC.pdf, 2021.
[255] SK Hynix, "H5ANAG8NCJR-XNC," 2021. [Online]. Available: https://www.digikey.com/en/products/detail/netlist-inc/H5ANAG8NCJR-XNC/20841590
[256] ——, "DDR4 SDRAM UDIMM Based on 8Gb D-die HMA851U6DJR6N HMA81GU6DJR8N HMA81GU7DJR8N HMA82GU6DJR8N HMA82GU7DJR8N," https://www.digchip.com/datasheets/parts/datasheet/2/202/HMA81GU7DJR8N-pdf.php, 2020.
[257] ——, "H5AN8G8NDJR-WMC," https://www.preduo.com/product/dram/ddr4/h5an8g8ndjr-wmc.
[258] Kingston, "4GB 1Rx8 512M x 64-Bit PC4-2133 CL15 260-Pin SODIMM," https://www.kingston.com/dataSheets/KVR21S15S8_4.pdf, 2015.
[259] Micron, "DDR4 SDRAM MT40A1G4 MT40A512M8 MT40A256M16," https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2583/MT40A1G4%2C%20512M8%2C%20256M16.pdf, 2015.
[260] ——, "DDR4 SDRAM SODIMM Addendum," https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/modules/sodimm/ddr4/atf4c1gx64hz.pdf, 2019.
[261] ——, "MT40A1G16KD-062E:E," 2020. [Online]. Available: https://www.farnell.com/datasheets/3151188.pdf
[262] ——, "MTA18ASF4G72HZ-3G2F1 Data Sheet," https://www.micron.com/content/dam/micron/global/secure/products/data-sheet/modules/sodimm/ddr4/asf18c4gx72hz.pdf, 2015.
[263] ——, "MT40A2G8SA-062E:F," 2021. [Online]. Available: https://www.arrow.com/en/products/mt40a2g8sa-062e-itf/micron-technology
[264] Kingston, "KSM32ES8/8MR 8GB 1Rx8 1G x 72-Bit PC4-3200 CL22 288-Pin DIMM," https://www.kingston.com/datasheets/KSM32ES8_8MR.pdf, 2021.
[265] Micron, "MT40A1G8SA-062E:R Data Sheet," https://www.micron.com/content/dam/micron/global/secure/products/data-sheet/dram/ddr4/8gb-ddr4-sdram.pdf, 2015.
[266] Samsung, "Unbuffered DIMM M378A2G43AB3-CWE," https://semiconductor.samsung.com/dram/module/udimm/m378a2g43ab3-cwe/.
[267] ——, "K4AAG085WA-BCWE." [Online]. Available: https://semiconductor.samsung.com/dram/ddr/ddr4/k4aag085wa-bcwe/
[268] ——, "Error correction code UDIMM / SODIMM M391A2G43BB2-CWE," https://semiconductor.samsung.com/dram/module/ecc-udimm-ecc-sodimm/m391a2g43bb2-cwe/.
[269] ——, "Small outline DIMM M471A5244CB0-CRC," https://semiconductor.samsung.com/dram/module/sodimm/m471a5244cb0-crc/.
[270] ——, "Small outline DIMM M471A4G43CB1-CWE," https://semiconductor.samsung.com/dram/module/sodimm/m471a4g43cb1-cwe/.
[271] Micron, "DDR4 SDRAM SODIMM Addendum," https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/modules/sodimm/ddr4/atf4c1gx64hz.pdf, 2019.
[272] ——, "MT40A1G16RC-062E:B," 2020. [Online]. Available: https://www.micron.com/products/memory/dram-components/ddr4-sdram/part-catalog/part-detail/mt40a1g16rc-062e-b
[273] Kingston, "KVR24N17S8/8 8GB 1Rx8 1G x 64-Bit PC4-2400 CL17 288-Pin DIMM," https://www.kingston.com/datasheets/KVR24N17S8_8.pdf, 2016.

A. Tested DRAM Modules

Table 2 shows the characteristics of the DDR4 DRAM modules we test and analyze. We provide the module and chip identifiers, manufacturing date (Mfr. Date), chip density (Chip Den.), die revision (Die Rev.), chip organization (Chip Org.), and the subarray size of tested DRAM modules. We report the manufacturing date of these modules in the form of $week - year$. Table 2 shows the minimum and average HC_{first} values for double-sided RowHammer, CoMRA, and SiMRA across all tested rows.

Table 2: Characteristics of the tested DDR4 DRAM modules.

Module Vendor	Chip Vendor	Module Identifier Chip Identifier	#Modules (#Chips)	Mfr. Date ww-yy	Chip Den.	Die Rev.	Chip Org.	Minimum (Average) HC_{first} RowHammer	CoMRA	SiMRA
TimeTec	SK Hynix	75TT21NUS1R8-4 [250] H5AN4G8NAFR-TFC [251]	1 (8)	N/A	4Gb	A	×8	38.45K (112K)	447 (5.84K)	585 (6.62K)
SK Hynix	SK Hynix	HMA81GU7AFR8N-UH [252] H5AN8G8NAFR-UHC [253]	8 (64)	43-18	8Gb	A	×8	25.0K (63.24K)	1885 (45.28K)	26 (16.14K)
Kingston	SK Hynix	KSM26ES8/16HC [254] H5ANAG8NCJR-XNC [255]	2 (16)	52-23	16Gb	C	×8	6.25K (17.13K)	4.54K (12.27K)	48 (16.02K)
SK Hynix	SK Hynix	HMA81GU7DJR8N-WM [256] H5AN8G8NDJR-WMC [257]	6 (48)	N/A	8Gb	D	×8	7.58K (23.11K)	632 (16.42K)	95 (22.81K)
Kingston	Micron	KVR21S15S8/4 [258] MT40A512M8RH-083E:B [259]	1 (8)	12-17	4Gb	B	×8	126K (338K)	93K (295K)	N/A
Micron	Micron	MTA4ATF1G64HZ-3G2E1 [260] MT40A1G16KD-062E:E [261]	4 (32)	46-20	16Gb	E	×16	4.89K (10.01K)	3.72K (7.69K)	N/A
Micron	Micron	MTA18ASF4G72HZ-3G2F1 [262] MT40A2G8SA-062E:F [263]	4 (32)	37-22	16Gb	F	×8	4123 (9.03K)	3.49K (7.06K)	N/A
Micron	Micron	KSM32ES8/8MR [264] MT40A1G8SA-062E:R [265]	2 (16)	12-24	8Gb	R	×8	3.84K (9.32K)	3.67K (7.67K)	N/A
Samsung	Samsung	M378A2G43AB3-CWE [266] K4AAG085WA-BCWE [267]	1 (8)	12-22	16Gb	A	×8	6.70K (14.80K)	5.26K (10.61K)	N/A
Samsung	Samsung	M391A2G43BB2-CWE [268] Unknown	5 (40)	15-23	16Gb	B	×8	6.15K (14.79K)	1875 (10.64K)	N/A
Samsung	Samsung	M471A5244CB0-CRC [269] Unknown	1 (4)	19-19	4Gb	C	×16	8.94K (25.83K)	6.25K (18.40K)	N/A
Samsung	Samsung	M471A4G43CB1-CWE [270] Unknown	1 (8)	08-24	16Gb	C	×8	6.81K (15.22K)	4433 (10.95K)	N/A
Samsung	Samsung	MTA4ATF1G64HZ-3G2B2 [271] MT40A1G16RC-062E:B [272]	1 (8)	08-17	4Gb	E	×8	15.77K (81.03K)	11.72K (60.83K)	N/A
Kingston	Nanya	KVR24N17S8/8 [273] Unknown	3 (24)	46-20	8Gb	C	×8	31.29K (128K)	20.19K (107K)	N/A

B. Discussion

PuDHammer on LPDDRx/DDR5. We believe the fundamental observations of PuDHammer likely apply to LPDDRx/DDR5 as well (since the DRAM array essentially has the same organization as in DDR4). Unfortunately, conducting LPDDRx/DDR5 experiments is extremely difficult since there is no robust open-source LPDDRx/DDR5 testing infrastructure. As a result, we do not know 1) if we can perform SiMRA and/or CoMRA in COTS LPDDRx/DDR5 chips and 2) how *severe* the read disturbance effects of SiMRA and CoMRA are in COTS LPDDRx/DDR5 chips.

Effect of DDR5's Smaller Refresh Window. Assuming 1) we can perform SiMRA and CoMRA operations in real DDR5 chips and 2) SiMRA and CoMRA's read disturbance effects in DDR5 and DDR4 are equally severe, DDR5's smaller refresh window (32 ms) would *not* prevent SiMRA or CoMRA bitflips. This is because the lowest HC_{first} observed for SiMRA (CoMRA) is 26 (447). Performing 26 (447) SiMRA (CoMRA) operations take 1.48μs (42.24μs), a very small fraction of the 32 ms DDR5 refresh window.