HammerSim: A System-Level Tool to Model RowHammer

Kaustav Goswami, Ayaz Akram, Hari Venugopalan, and Jason Lowe-Power

University of California, Davis

Abstract

Modern architecture research relies on simulators to evaluate system security, yet analyzing emerging hardware vulnerabilities like RowHammer requires full-system visibility. As RowHammer vulnerabilities worsen with continuous technology scaling, existing simulators lack the system-level models needed to study complex OS effects and cross-layer mitigations. This tool deficiency leaves modern computing platforms exposed to severe reliability and security risks. In this work, we present HammerSim, a gem5-based framework for modeling RowHammer at the full-system level. HammerSim integrates probability-driven bitflip modeling to realistically capture the behavior of RowHammer. It further enables evaluation of hardware and software mitigations such as TRR and selective ECC. We validate HammerSim's bitflip modeling against real DDR4 DIMMs using JS divergence, demonstrating its utility in studying attacks, defenses, and benign workload susceptibility. Our framework provides an extensible platform to bridge the gap between hardware experiments and architectural simulation.

1. Introduction

Dynamic random access memory, or DRAM, is the *de-facto* choice for main memory design due to its cost-effectiveness. However, due to scaling, researchers have noticed variations in the nominal parameters of these devices [9, 99, 50]. Recently, it has been found that accessing (or ACTivating) the same cell or the same set of cells on a DRAM module repeatedly causes data corruption in the neighboring cells. This is called RowHammer [5, 43]. Researchers have shown several system-level attacks using RowHammer to escalate privileges, fingerprint or leak private data [47, 92, 82, 8, 83, 67, 27, 90, 3, 10, 13, 12, 19, 20, 26, 35].

Currently, researchers rely on two primary hardware-based methods to study RowHammer: *(a)* FPGAs, and *(b)* software-based exploits [69, 68, 87, 1]. FPGA-based memory controllers [31, 63, 65] (SoftMC [31], DRAM-Bender [65], *etc.*) provide highly accurate timing and have been instrumental in uncovering proprietary in-DRAM mitigations like Target Row Refresh (TRR) [30]. However, FPGAs lack the capability to model OS-level abstractions or full server-class system dynamics. Conversely, software-based RowHammer exploits [20, 36, 14, 90, 41, 27, 91] successfully bypass OS-level protections and allow researchers to explore kernel-level vulnerabilities [3, 96]. Yet, they do not provide the hardware visibility necessary to prototype or evaluate new hardware-software co-design mitigations.

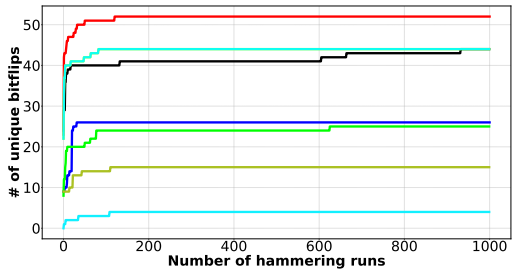


Figure 1: Count of unique bitflips on different DRAM DIMMs' row. Each line plot corresponds to a victim row. In total, we collected victim row data from 9 different DIMMs from two different manufacturers.

Researchers today use cycle-level simulations [43, 97, 86, 70, 46, 25, 84] to bridge this gap between the hardware-level and software-level view of RowHammer. However, popular cycle-accurate and cycle-level simulators [78, 48, 44, 52, 74] used for memory system research today assume an idealistic setting, often using timing, current, and voltage values from data sheets for simulating hardware components [93, 78, 48, 44, 74, 52]. To model variations, researchers have to explicitly add variations [22, 99, 85], which generally leads to over/under-estimation of the simulation model. While some models of process variation exist purely as a statistical models [81, 2, 85], we do not have models for RowHammer. Further, the RowHammer threshold is decreasing as DRAMs are becoming denser [42], which motivates the need for such a model.

To understand the fidelity gap of such uniform models, we performed extensive RowHammer characterization experiments across multiple real-world DRAM Dual Inline Memory Modules (DIMMs). As shown in Figure 1, our hardware evaluations reveal that RowHammer-induced bitflips are highly non-uniform; unique bitflips saturate over time, and certain memory regions exhibit significantly higher vulnerability than others.

Crucially, prior literature demonstrates that real-world RowHammer behavior is deeply intertwined with full-system parameters, including the operating system page allocation [10, 83, 36], CPU frequency [92], cache coherency protocols [50], instruction set architectures (ISAs) [43, 83, 20, 36, 3, 91, 55], and underlying memory technology [43, 46, 30]. Even benign applications running under normal system workloads have been shown to inadvertently trigger RowHammer bitflips [66]. Without the ability to simulate

how hardware faults dynamically interact with OS scheduling and cache hierarchies, researchers cannot reliably estimate security boundaries or system reliability.

To solve this, we propose HammerSim, a full-system RowHammer modeling framework built upon the gem5 architectural simulator [52]. HammerSim directly addresses the shortcomings of prior tools by combining the physical accuracy of hardware-characterization maps with the system-level visibility of an architectural simulator. While circuit-level models [40] or trace-driven simulators cannot natively capture live OS reactions, HammerSim runs an unmodified OS. It enables researchers to analyze realistic RowHammer data corruption under complex workloads, cache hierarchies, and OS scheduling, while actively supporting the evaluation of sophisticated hardware-software mitigation co-designs

Overall, the contributions of this paper are as follows:

- System-level RowHammer model in gem5 that incorporates multiple probability distributions for realistic bitflip estimation, enabling studies on real workloads and future DRAM technologies with or without mitigations.
- Dual analysis modes with data corruption support where an *online mode* enables in-situ data corruption during simulation to study system-level effects, and an *offline mode* provides post-processing for accurate probability estimation and correction of over/under-estimations.
- Hardware-informed modeling and validation where extensive RowHammer experiments on DDR4 DIMMs to capture real-world bitflip patterns and validate simulation fidelity using JS divergence.

The rest of the paper is organized as follows. Section 2 discusses the basics of a DRAM device, the RowHammer vulnerability and Section 3 describes our RowHammer model in a hollistic manner. Section 5 analyzes the simulations. The work in concluded in Section 7 with further scope of improvement.

2. Background

In this section, we introduce the basics of a DRAM device, how it is implemented as an interface in gem5, and the RowHammer vulnerability.

2.1. DRAM Organization

DRAM has spawned several generations and technologies. This includes DDRx [58, 59, 61, 39], GDDRx [33, 62], LPDDRx [57], DDRxL [60], *etc.* Each physical DRAM Dual Inline Memory Module or DRAM DIMM is installed on a DRAM *channel* on the motherboard. Channels permit issuing concurrent memory requests to multiple DIMMs. A DIMM contains multiple logical structures called *banks* on one or both of the *ranks*. A bank is a two-dimensional array of cells organized into *rows* and *columns*. Each cell contains a capacitor and an access transistor, with the capacitor's charged state representing a single bit. Organizing cells into banks enables faster memory access since memory addresses

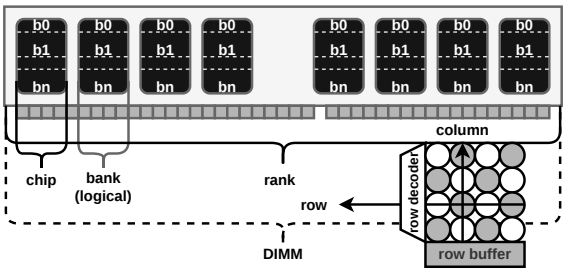


Figure 2: This figure shows a single rank of a DRAM DIMM. Each rank contains multiple logical structures called banks that are interspersed across multiple physical structures called chips.

that map to different banks can be accessed in parallel. Independent dies are packaged into multiple physical entities called *chips*. The banks on each rank are uniformly distributed across these chips (Figure 2).

2.2. Working of a DRAM Device

The memory controller receives memory requests with an instruction and a corresponding address to the DRAM. These addresses encode information about the DRAM structure (channel, rank, bank, etc). A mapping function uniquely maps addresses to a set of capacitors within the DRAM [72][1]. On receiving a memory request, the memory controller uses this function to identify the corresponding bank within the corresponding DIMM. The address is then pushed into the bank's *row-decoder* logic. Then the row-decoder selects the appropriate row and transfers it to a *row-buffer*, which is known as an `ACTIVATE`, or `ACT`. The row-buffer is an array of sense amplifiers that sense the charge of the cells in that row to determine their bit values. Finally, the bank's *column-decoder* reads the values of the appropriate cells within the row-buffer and returns the corresponding data.

DRAM technology is based on capacitors, which inherently lose its charge over time [34]. Their contents have to be recharged periodically to preserve the integrity of the data stored. The memory controller issues a `REFRESH (REFI)` instruction to a subset of rows every 7.8 μs (`tREFI`) to preserve the values of their capacitors. To refresh a particular row, it is first activated to sense the bit values. Then, the bits are subsequently restored to the DRAM array. Every DRAM capacitor gets refreshed at least once every 32 ms or 64 ms (`tREFW`) based on the operating temperature.

2.3. gem5's Memory Controller and Memory Interface

The memory controller in gem5 is simulated as an event-driven component [28]. It models a generic memory

1. Note that this is not a virtual-to-physical address mapping. The virtual address is translated to its corresponding physical address in the Translation Lookaside Buffer (TLB) [71]. This mapping is physical to device mapping.

controller with split queues for reads and writes, and a shared queue for responses. It is responsible for managing the communication between the processor and the memory. gem5 receives memory requests from the processor and schedules these requests based on a given scheduling policy. The memory organization it models is similar to its regular counterpart. Addresses decode into rank, bank, row, and column at each memory controller. The channel interleaving is decoded outside at a separate crossbar. A Packet is used to encapsulate a transfer between two objects in the memory system [52]. An LLC miss in gem5 is seen as a packet coming into the memory controller. Once the packet is received at the memory controller, it decodes that packet, schedules it, and sends it to the memory interface.

Memory devices are modeled as classes in gem5. Currently in gem5, there are two memory interfaces: (a) DRAM and (b) NVM. The timing parameters for these devices are based on the technical sheet of these devices. Hansson *et al.* [28] mentions that gem5 does not model all the timing parameters from a contemporary memory device. However, gem5 models all the important timing parameters, which were validated against a cycle-accurate DRAM simulator.

2.4. The RowHammer Vulnerability

Scaling of DRAM devices over the generations has resulted in the deviation of their nominal parameters from their theoretical counterpart due to variations [99, 9, 82]. Modern DIMMs are susceptible to memory corruption as a result of electrical interference among the cells [53]. One such technique to leverage bit flips without accessing the row is the rowhammer (RH) exploit [5, 43]. It causes memory corruption by repeatedly accessing two addresses that map to two different rows in a single bank. This leads to the repeated activation and deactivation of the two rows back-and-forth between the row-buffer and the DRAM bank array. The resulting electro-magnetic interference between the accessed rows (referred to as *aggressor rows* henceforth) and their neighboring rows (referred to as *victim rows* henceforth) accelerates the rate at which the capacitors in the victim rows lose their charge. Once capacitors have lost a sufficient amount of charge, refreshing the DRAM does not restore their value, resulting in memory corruption in the form of bit flips.

There have been several works done in the past to detect and mitigate RH attacks [43, 86, 84, 46, 70, 97]. DRAM manufacturers implement TRR added the standard mitigation mechanism in DDR4 devices as a practical solution that does not violate DRAM timings [80, 61, 20].

3. HammerSim

Our objective is to develop a comprehensive, full-system simulation framework that accurately mirrors the hardware-level probabilistic and spatial characteristics of RowHammer. In this section, we list the challenges, perform extensive hardware exploration, incorporate the findings and discuss the implementation details.

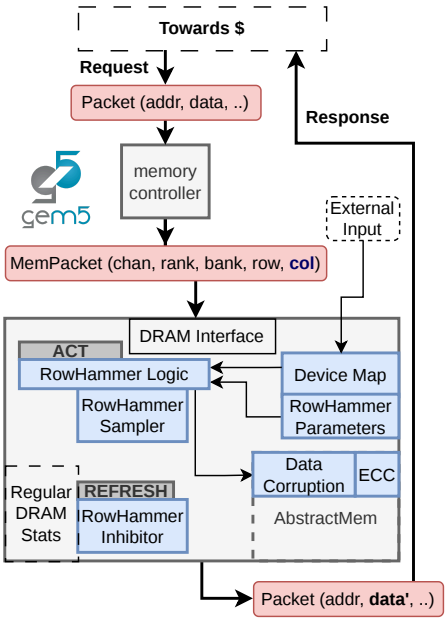


Figure 3: High-level overview of RowHammer Modeling in gem5. HammerSim architectural components are highlighted in blue. The core modeling logic is integrated within the gem5 DRAM interface, while the execution path is configured via flexible user parameters.

3.1. Architectural and Simulation Challenges

Standard circuit-level or trace-driven memory simulators cannot resolve the three fundamental design challenges introduced when designing a scalable, full-system simulation infrastructure for RowHammer:

C1 **Representing Non-Uniform Hardware Non-Idealities.** Traditional architectural simulators treat faults as uniform or binary events. However, real-world hardware data reveals that RowHammer bitflips are heavily non-uniform and highly localized due to manufacturing process variations. Translating these irregular physical vulnerabilities into deterministic, flexible, and mathematically sound probability models within an architectural simulator is highly non-trivial.

C2 **Spatial Tracking Overhead at Scale.** To model multi-row vulnerabilities (e.g., Half-Double [23, 45] or multi-sided attacks [43, 20]), the memory controller must continuously maintain and update activation tracking structures across spatial neighborhoods. Performing these multi-row spatial lookups at runtime for every memory access introduces severe simulation timing and memory overheads, requiring highly optimized metadata tracking.

C3 **Balancing Execution Fidelity with Multi-Parameter Exploration.** To evaluate hardware-software defenses,

TABLE 1: Experimental System Parameters

Parameter	Specification
CPU type	Intel(R) Core(TM) i7-7700 CPU
CPU core count	4
Frequency	3.6 GHz
L1 cache per core	32 KiB + 32 KiB (I + D)
L2 cache	1 MiB
L3 cache	8 MiB
DRAM manufacturers	SK Hynix, Samsung
DRAM technology	DDR4
DRAM size	8 GB

researchers require an *online mode* capable of dynamically corrupting live memory data to capture operating system reactions and application-level crashes. Conversely, security researchers exploring different hardware topologies require rapid, trace-driven sensitivity sweeps. The framework must seamlessly decouple full-system execution from trace-driven analytical processing without duplicating simulator infrastructure.

3.2. Hardware Characterization Experiments

To ground our architectural probability models in physical ground truth, we conducted extensive hardware characterization experiments across multiple commercially available DDR4 DIMMs. The goal of these experiments is to quantify the exact spatial and temporal probability distributions governing RowHammer failures under controlled environments.

We deployed the Blacksmith [36, 37] profiling infrastructure on a desktop detailed in Table 1. We executed double-sided RowHammer profiling by targeting a specific victim row sandwiched between two continuously activated aggressor rows. Each profiling loop executed 5×10^6 `ACTs`, and the routine was repeated over 1,000 distinct iterations to observe long-term bitflip patterns, cumulative error saturation, and localized vulnerability profiles.

3.3. Empirical Observations

Our physical characterization yielded three core insights regarding the nature of modern DRAM vulnerabilities, which serve as the direct mathematical foundations for the HammerSim architectural abstraction layers:

(a) Run 1 (b) Run 2 (c) Run 3 (d) Run 4 (e) Superimposed Image

Figure 4: Observed bitflips on a single row on a width 8 DRAM DIMM (row 1278). There are 1024 columns per row. Each of these images presents a 32x32 shaped 2D image of a single row. A white dot represents a flipped column.

Observation 1 (Spatial Localization): RowHammer-induced bitflips are intensely localized within the memory array. While specific regions of a given DRAM row are highly vulnerable to RowHammer, vast adjacent structures remain entirely resilient.

This behavior (Figure 4) indicates that local manufacturing process variations dictate cell weakness. Rather than hard-coding these patterns for a single device, HammerSim resolves *C1* by providing two generalized modeling paths: an ingested, file-based physical `device map`, or a statistical process variation engine based on VARIUS [81] that generates multivariate normal distributions across the simulated memory coordinates to flag distinct *Weak Cells* (`WC`).

Observation 2 (Temporal Saturation): The unique bitflip count across a DRAM array saturates over time. Cells that fail during an initial activation cycle demonstrate a bounded, distinct probability p of failing again across subsequent cycles.

The empirical histograms in Figure 5 shows this saturation behavior. This establishes that the pool of vulnerable cells is structurally finite. Once a simulated cell is identified as a `WC`, HammerSim evaluates a uniform probability distribution at runtime to determine if an active activation stresses the cell into a flippable state (`F_WC`). We measured this baseline bitflip probability between $\frac{1}{5\times10^{10}}$ and $\frac{1}{5\times10^{8}}$ per activation, which we expose as a user-tunable parameter.

Observation 3 (Attack Proximity Non-Linearity): The probability of inducing a data failure scales non-linearly by multiple orders of magnitude when executing coordinated multi-sided RowHammer patterns relative to single-sided access sequences.

Our physical tracking demonstrated that executing a double-sided RowHammer sequence amplifies the bitflip probability by a factor of 2.5×10^3 compared to a single-

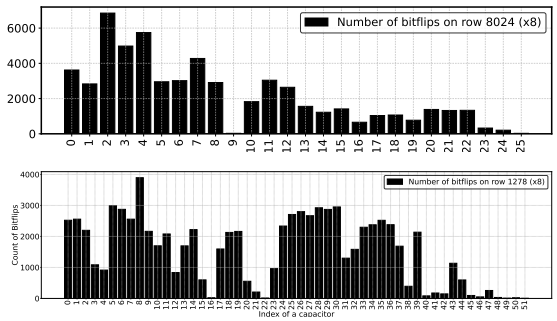


Figure 5: Histogram of bitflip counts on row number 8024 (top) and 1278 (bottom) across 1000 runs.

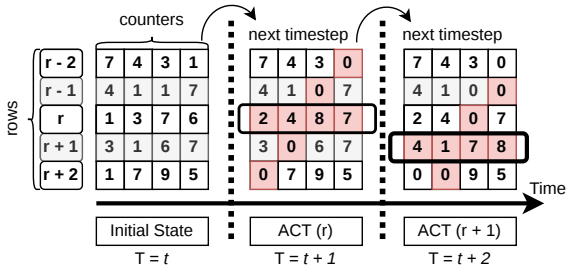


Figure 6: HammerSim maintains structural counter-pairs per DRAM row to to dynamically track multi-row RowHammer access patterns. All counters are reset at each `tREFW` boundary. This figure shows 3 states of these counters: initial, after reading row *r* and after reading row *r + 1*.

sided baseline hitting the identical weak cell cluster. To implement this insight, HammerSim leverages dynamic tracking counters inside the memory interface, tracking the nonlinear threat scaling of single-sided, double-sided, and generalized multi-row patterns.

3.4. Architectural Implementation

To incorporate these empirical insights into a unified simulation framework, HammerSim introduces structural and systemic modifications to the gem5 DRAM interface, implementing both online and offline execution modalities.

3.4.1. Counter-Pairs and Adjacency Tracking. To resolve *C2* and capture complex spatial interactions without exponential simulation overheads, HammerSim maintains optimized tracking structures within the memory controller. As illustrated in Figure 6, each row tracks its immediate neighborhood using counter-pairs. For standard vulnerabilities, this mechanism monitors single and double-sided access rates.

To support advanced vulnerabilities like Half-Double exploits [23], we expand the tracking radius to monitor four dedicated row activation counters ($r \pm 1, r \pm 2$) relative to the target row r. To optimize performance, these structures evaluate fault triggers during standard refresh intervals (`tREFW`), clearing all tracking metadata at each window boundary.

3.4.2. Online Mode: Runtime Fault Injection and Functional ECC. In *online mode*, HammerSim couples its hardware-informed probability distributions directly into gem5's operational memory execution path. Every memory request dynamically triggers fault evaluation checks based on current row activation states, injecting live, bit-level data corruption into the simulator's active memory arrays.

Because physical RowHammer bitflips occur silently within the DRAM capacitors without throwing hardware interrupts, they act as transient soft errors. HammerSim replicates this behavior: while all injected errors are logged to standard output (`stdout`) for diagnostic tracking, the simulated application and operating system remain unaware of the corruption until they actively address or validate the corrupted memory address. This allows for authentic evaluation of silent data corruption across OS-level page schedulers, security kernel modules, and user space applications.

To support the exploration of cross-layer resilience, we implement a functional Error Correcting Code (ECC) abstraction layer within gem5's memory subsystem. ECC-enabled DIMMs [88] are a popular option for server-class systems [13]. RowHammer on ECC-enabled DIMMs is a popular topic of research for both attacks [13] and defenses [54, 95, 76].

We implement SECDED algorithm from the reverse-engineered paper by Cojocar *et al.* [13]. HammerSim pads every word with parity blocks calculated via a user-defined text evaluation matrix (`pMatrix`). At runtime, when a corrupted row is accessed, the ECC module intercepts the request, evaluates the parity matrix, and dynamically updates native gem5 telemetry statistics (`stats.txt`) to record correctable single-bit faults or fatal, uncorrectable multi-bit errors.

3.4.3. Offline Mode: Decoupled Trace Analysis. To address *C3* and enable rapid design space exploration without the penalty of repetitive, long-running full-system simulations, HammerSim introduces an optimized *offline mode*. When active, the memory controller logs row-level activation (`ACT`) allocations to a structured trace file at every `tREFW` boundary.

An external analytical tool subsequently processes the RowHammer trace that overlays the workload's spatial activation footprint against alternative process variation distributions, device maps, or mitigation thresholds. This decoupling allows hardware designers and security analysts to evaluate how an identical application execution profile interacts with dozens of distinct DRAM manufacturing variations instantly, eliminating the need to re-run the underlying architectural simulation.

4. Discussion and Limitations

4.1. RowHammer Research Enabled

HammerSim allows researchers to investigate RowHammer attacks, defenses, and, also potentially study applications of RowHammer. In this section, we discuss several use cases that HammerSim enables.

4.1.1. RowHammer Probability Distribution Study. HammerSim allows researchers to investigate different probability distributions of causing a bitflip. We incorporate known probabilities from prior works [92, 43, 20, 82, 23, 45] and our experimental observations. However, there is no mathematical evidence that these are all the distributions that define RowHammer. The probability of a bitflip of a strong DRAM cell, for example, is still missing from our evaluation. HammerSim can be extended in the future to incorporate such distributions, when discovered, and studied in a full-system environment.

4.1.2. RowHammer Attack. HammerSim allows users to uncover sophisticated memory access patterns that cause RowHammer attacks. Creating a RowHammer attack as a memory trace is straightforward by using gem5 traffic generators. Such traffic generators produce deterministic access patterns for a given device map. Various RowHammer attacks like Juggernaut attack pattern [94], double RowHammer attack [25], *etc.*, can be created using HammerSim. These maps can be refined by researchers or vendors using process variation data.

In contrast, implementing RowHammer at the system level is far more complex. Prior attacks [10, 83, 3, 20, 36, 41, 14, 27] show that bypassing OS abstractions is challenging. While evicting caches suffices for single-sided RowHammer on x86 systems, obtaining contiguous rows on the same bank in multi-channel, multi-rank servers remains difficult [92]. Furthermore, physical address mapping strongly influences RowHammer bitflip behavior [72, 20].

Further, HammerSim can model GPU-based RowHammer attacks [91, 49, 73, 32] using the GPU model in gem5 [75, 77].

4.1.3. RowHammer Mitigation. HammerSim enables users to prototype and evaluate hardware-based RowHammer mitigations at the system level. It supports two key components: (a) detection and (b) inhibition, modeled with timing correctness. To demonstrate this flexibility, HammerSim replicates simplified TRR mechanisms based on reverse-engineered designs [30, 36] from major DRAM vendors, including counter-based and probabilistic refresh strategies. These implementations show how researchers can extend HammerSim's data structures to explore new mitigation techniques without breaking DRAM timing constraints.

TRR refreshes are sent out when the DRAM DIMM is locked (for tRFC time) during refreshes (via REFI instructions). Most research-based RowHammer mitigations, on the other hand, explicitly send out refreshes as `ACTIVATE` to the victim rows, violating DRAM timings. DRAM devices adhere to strict timing requirements [59, 61, 80, 39, 33, 62]. TRR [80, 20, 36] sends targeted REF commands [20] (or simply `ACT` commands) to possible victim rows.

HammerSim allows simple extensions to the existing data structures to implement a RowHammer mitigation techniques like [43, 46, 70, 29, 97, 84]. Further, HammerSim allows tweaking such mechanisms to adhere to the timing contraints set by existing TRR mechanisms via the Inhibitor module. To show that, we implement some of the classic and *state-of-the-art* RowHammer mitigations like PARA [43] and TWICE [46].

4.1.4. Error Correcting Code. Another direction of RowHammer mitigation is ECC [13, 54, 76, 95]. ECC-enabled DIMMs are commonly available in the market today. DDR5 DIMMs are shipped out with an on-chip proprietary ECC implementation [38, 56].

Qureshi *et al.* suggested investigating RowHammer defenses via newer ECC algorithms [76]. Online HammerSim enables functional ECC modeling to study RowHammer defenses in a full-system context. It supports selective error correction during simulation, allowing researchers to evaluate ECC schemes under realistic workloads. For demonstration, we implement a SECDED algorithm (1-bit correction, 2-bit detection) using Cojocar *et al.* reverse engineered ECC algorithm [13] on a DDR3 DIMM within HammerSim.

ECC bits are computed from a user-supplied `pMatrix` and applied on-demand to minimize simulation overhead. This capability allows exploration of new ECC strategies and their effectiveness against RowHammer-induced corruption.

4.1.5. RowHammer Applications. HammerSim enables researchers to explore RowHammer-based applications such as PUFs [82, 98] and DRAM fingerprinting [92, 47] by modeling cell-level variation maps in simulation. Unlike prior simulators that assume uniform vulnerability, HammerSim uses device maps derived from real DIMMs or statistical models to capture per-cell variation. This capability also supports evaluating defenses like row-swapping [79, 94, 24] by identifying strong and weak cells, reducing overhead while preserving security. Further, in Section 5.3.2, we show how HammerSim can be used to study RowHammer-based soft-errors [15].

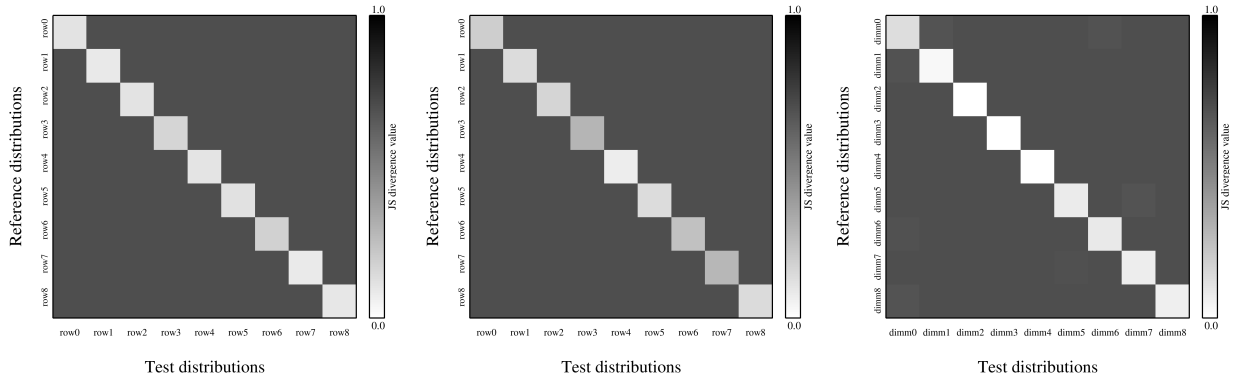
4.2. Limitations

HammerSim introduces invasive changes to gem5's memory model by embedding RowHammer tracking and mitigation logic directly in the memory controller interface, rather than using separate SimObjects. This design mirrors real DRAM behavior, where TRR operates on-chip. However, this approach requires periodic updates and rebasing as gem5 evolves. Making changes to the source is intuitive as our RowHammer trackers get triggered when `activateBank()` is called and the inhibitor is called at `processRefreshEvent()`, similar to the on-chip logic of a typical DIMM [20].

A key limitation is the abstraction gap: simulators [78, 48, 44] model DRAM at the logical level (banks and cache lines), whereas physical DIMMs activate entire rows spanning multiple chips. HammerSim currently applies bitflips within the same logical bank as the aggressor, while ensuring the correctness of the victim rows. This means that the data is tracked at the cache line level using simulator-dependent data objects.

Another limitation is reliance on statistically generated variation maps. While multivariate models approximate process variation and are popular [99], they may deviate from real DIMM characteristics. To address this, HammerSim supports user-supplied device maps derived from hardware experiments (e.g., Blacksmith [36], TRRespass [20]).

ECC is functionally implemented in HammerSim. Unlike a real ECC DIMM that stores the checksum bits on the DRAM array, we compute the checksum on the fly to model functional correctness. This allows newer ECC algorithms to be evaluated online during simulation. However, checksum



(a) JS divergence heatmap of 9 different rows on dimm0

(b) JS divergence heatmap of 9 different rows on dimm1

(c) JS divergence heatmap of 9 different DIMMs from the same DRAM manufacturer hammering row 7292

Figure 7: JS divergence heatmaps across different victim rows of the same DIMM and across multiple DIMMs on real hardware and HammerSim.

data corruption and timing correctness of the memory are not modeled.

Finally, HammerSim does not model true and anti cells [43], as these mappings are proprietary. A charged DRAM capacitor does not necessarily mean a value of one [43, 36, 20]. Extending HammerSim to enable such mappings is trivial and can be implemented the same way as the device map. This does not affect the RowHammer probabilities of a bitflip.

5. Evaluation

HammerSim is implemented in gem5 [52], a full-system cycle-level simulator. We model the probability distributions of RowHammer (Section 3) within gem5's memory interface.

HammerSim allows users to perform several kinds of experiments. We narrow down the search space and perform three case studies to evaluate HammerSim. We fix our experimental system as a commodity-class system in most of our experiments. The parameters of the system are described in Table 2.

We validate our simulation model against real hardware in Section 5.1. In Section 5.2, we show how a full system RowHammer application can be studied in HammerSim and how ECC can functionally correct bitflips. Finally, we also demonstrate how benign applications are affected by RowHammer as the RowHammer threshold is decreasing with each DRAM generation.

5.1. Case Study I: Validation

We want HammerSim to model existing DIMMs inside the simulator. This implies that we also need to model TRRs installed in the DIMMs. Towards this, we have implemented RNG-based TRR mitigation techniques in HammerSim, conceptually similar to U-TRR [30]. We use Blacksmith [36]'s fuzzer to find access patterns on the same

TABLE 2: Simulated System Parameters

Parameter	Specification
CPU type	TimingSimpleCPU
CPU core count	1/2/8
L1 cache per core	32 KiB + 32 KiB (I + D)
L2 cache	256 KiB
DRAM technology	DDR3/DDR4
DRAM size	1/2 GB

DIMMs. Bitflips are generated per bank. We record the aggressor traffic. We encode the bitflip maps generated from the hardware and plug in our simulation as the variation map.

Our goal in this case study is to model a similar bitflip map in simulation. We split the test cases into within rows of a single DIMM and within banks of multiple DIMMs. Blacksmith's fuzzer bypasses TRR with a specific memory access pattern. We use gem5's traffic generator to generate synthetic traffic for the aggressor rows, recreating the attack. gem5's traffic generator can send a large amount of traffic at the minimum interval of 1 ps. We adjust our traffic pattern to mimic real systems with caches and tune the RowHammer probabilities to account for such high traffic rates. Note that the device address mapping scheme, *i.e.*, physical to the DRAM device mapping, is different in the experimental system and the simulated system. We make sure that the reference and the test systems have compatible mapping.

We compare the weak columns of hardware-generated bitflips with the simulated ones for single rows. We use JS Divergence [64] as the similarity metric to compare bitflip maps. JS divergence is a method of measuring the similarity between two probability distributions. A lower JS divergence value means the two distributions are more similar. We present our results as a heatmap of JS divergence values with references collected from the hardware and tested using HammerSim. We select 9 rows from two different DIMMs

from the same DRAM manufacturer and show our findings in Figure 7(a) and Figure 7(b). We see smaller JS divergence values for the same reference and test case, suggesting a closer to hardware model of RowHammer in HammerSim.

Next, we repeat the same experiment across multiple DIMMs. We select 9 different DIMMs from the same manufacturer and model the RowHammer device map in HammerSim. We plot the heatmap of JS divergence values in Figure 7(c), which shows a close resemblance to the real hardware.

Takeaway 1: HammerSim allows localized RowHammer bitflip simulation via the device map. Weak cells per row can be modeled with high fidelity in HammerSim, up to the capacitor level, given that TRR does not interfere. While a correct reverse-engineered TRR implementation is not the goal of this paper, we show how HammerSim can be used to model the entire DIMM in simulation for higher correctness.

5.2. Case Study II: Full-System Applications and Data Corruption

We show the execution of Google's `rowhammer-test` [83] inside the simulator in a full-system setting. The OS abstractions are bypassed by rowhammer-test, as it can perform both single-sided and double-sided RowHammer attacks. Our objective is to test whether we can use rowhammer-test as a standard RowHammer benchmark workload.

We simulated `rowhammer-test` inside HammerSim using a reproducible gem5-resource [11] disk-image. We hammered the memory using the single-sided RowHammer attacks from `rowhammer-test` on a system specified in Table 2. We ran a simulation of one iteration of `rowhammer-test`. We show the bitflip count in Figure 8 by varying the probability of a bitflip. `rowhammer-test`'s output is printed on `stdout` with the number of bitflips and the change in data, as shown in Figure 9.

As mentioned in Section 4.2, we do not model true and anti-DRAM cells. For a bitflip, we flip the contents of the capacitor regardless of a charged or an unchanged capacitor. `rowhammer-test` writes `0xFF` as the data pattern and compares it for bitflips later. Bitflips are XOR on the exact capacitor location.

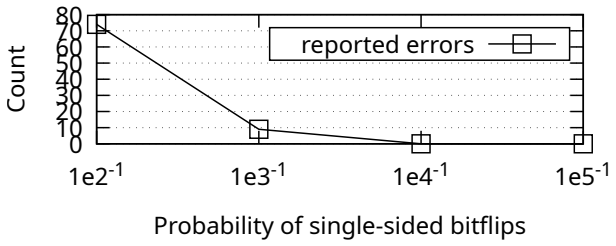


Figure 8: Reported bit errors by the simulator.

```
Starting gem5 init... trying to read run script file via readfile.
Running script from gem5-bridge stored in /tmp/script
rowhammer_test
[sudo] password for gem5: clear
cloud-init[1290]: Cloud-init v. 23.1.2-0ubuntu0~22.04.1 finished at Tue,
ource DataSourceNone.  Up 43.50 seconds
cloud-init[1290]: 2025-12-02 18:47:57,632 - cc_final_message.py[WARNING]:
Iteration 0 (after 0.01s)
  Took 338.2 ms per address set
  Took 3.38249 sec in total for 10 address sets
  Took 78.298 nanosec per memory access (for 43200000 memory accesses)
  This gives 102173 accesses per address per 64 ms refresh period
error at 0x7fae5ab6d1f0: got 0xffffffffffffffef
error at 0x7fae5bdbf150: got 0xfeffffffffffffff
error at 0x7fae5bdc31d0: got 0xffffffff7fffffff
error at 0x7fae5bdc3200: got 0xffffffffdfffffff
error at 0x7fae5bdc32a8: got 0xffdfffffffffffff
error at 0x7fae5bdc33c8: got 0xfeffffffffffffff
error at 0x7fae5e2cf008: got 0xffffffff7fffffff
```

Figure 9: `stdout` of `rowhammer-test` when the *single-sided* RowHammer attack was performed online using HammerSim.

Takeaway 2: HammerSim enables simulated data corruption shown using `rowhammer-test`. This allows system-level RowHammer attacks, defenses, and applications to be studied in simulation.

We extend the previous case study to test ECC within the simulator. We functionally model ECC, that does not account for timing changes in the DRAM. At a bitflip, we store the original data for ECC parity bit computation. When victim rows are later accessed, we call the ECC algorithm to keep the wall clock time of the simulation low. Whenever RowHammer triggers a bitflip, we compute the `gMatrix` based on an 8-byte word and the user-supplied `pMatrix`.

For single-bit errors on an 8-byte word, we use the ECC bits to correct the data when the victim row is accessed. HammerSim reports the bitflips as an output statistic. We keep the simulation parameters the same as before. As we increase the probability of a bitflip, we see multiple bitflips on the same row (Table 3). The SECDED ECC corrects up to one bit flip and detects up to 2 bit flips [13].

Takeaway 3: Data corruption in HammerSim naturally allows the simulator to investigate newer and stronger error correction codes (ECC).

TABLE 3: SECDED ECC statistics of correcting RowHammer-induced bitflips as reported by HammerSim. Bitflips are generated randomly based on the user-supplied probability.

Probability	Bitflips	Single Errors Corrected	Double Errors Detected	Uncorrectable Errors	Notes
$1e1^{-1}$	237	29	2	17	Kernel panic
$1e2^{-1}$	115	3	0	1	None
$1e3^{-1}$	95	1	0	0	None
$1e4^{-1}$	0	0	0	0	None

5.3. Case Study III: Benign Applications

5.3.1. Offline Analysis. To demonstrate HammerSim's offline analysis, we dump the memory access traces of two popular benchmark suites: NAS-Parallel benchmarks [4] and GAPBS [6]. In our experiment, we did not see any RowHammer access patterns in these benign applications with the standard bitflip probability parameters. With shrinking process size, the RowHammer threshold is also decreasing [20, 43]. Therefore, we perform an offline estimation of bitflips on these memory traces.

HammerSim allows us to tune probabilities of bitflips to correct over/under estimations via the post-processing tool. We show this by varying the probability of observing a single-sided RowHammer-induced bitflip from 10^{-9} to 1. The results obtained are depicted in Figure 10(a) and Figure 10(b).

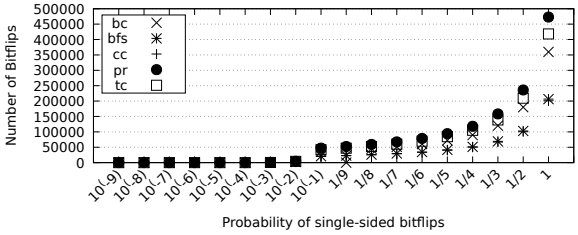
The memory access pattern for NPB and GAPBS never had double-sided RowHammer patterns. First, we experiment assuming a uniform distribution of observing bitflips and did not rely on the device map. We see that there are 20041.16 instances on average across the two benchmark suites, where rows cross the DDR4 threshold of 45K ACTs.

We repeat the same evaluation using a statistically generated device map for RowHammer bitflip distribution. The idea is to ignore *strong* DRAM rows, which are resistant to RowHammer attacks. We see that on average, there is a reduction of 90.1% probability of observing a bitflip compared to the previous set of results. Our observation inside a simulator is consistent with bitflip patterns observed in prior works [92, 91, 82, 21, 47], where authors of the respective papers report localized RowHammer-induced bitflips.

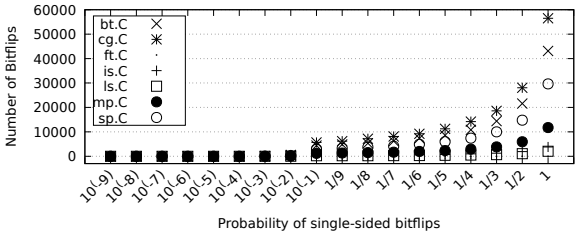
> **Takeaway 4**: This experiment shows that HammerSim can be used to investigate RowHammer access patterns at a finer granularity while enabling users to estimate RowHammer bitflip probability.

5.3.2. Online Analysis and Data Corruption Study. In our experiments, we did not see any instances of RowHammer bitflips with the default RowHammer parameters of a DDR4 DRAM DIMM. We repeated the experiment with a lower RowHammer threshold (1000) and a higher RowHammer probability ($1e1^{-1}$). However, we are still interested to observe soft errors in benign applications. Towards this, we repurpose HammerSim's infrastructure to study RowHammer-induced bitflips in the neighborhood of the aggressor row with a very low RowHammer threshold in benign applications. The goal is to observe silent error propagations on benign applications when the RowHammer threshold becomes dangerously low for future shrinking process.

We selected smaller workload sizes from our two selected benchmark suites (NPB class *A* and GAPBS size *-g 20*). The goal is to finish at least one iteration of ROI and validate the results on a system without caches to maximize memory accesses. We set up the experiment with



(a) Gapbs



(b) NAS parallel benchmarks

Figure 10: Offline estimation of bitflips by varying the probability of a bitflip for a DDR4 DRAM device. Note that the x-axis in both of these graphs are discontinous as it progresses in factors of 10 until (8, 0), and then it progresses in steps of 0.1.

data corruption enabled, set the RowHammer threshold (1), and the probability to study soft errors ($1e^{-1}$).

We model a system with a single CPU core without caches and 1 GiB DDR4 DRAM memory. For the baseline, we simulate an ideal DDR4 DRAM device without the possibility of any bitflips to measure the host system's execution time, total memory consumption, and output correctness. We repeat this experiment by replacing the DRAM with RowHammer-affected ones and simulate the same workload with and without data corruption.

Table 4 shows the simulation statistics of HammerSim and baseline gem5. Overall, we saw HammerSim takes 12.8% and 29.2% additional wall clock time on average to simulate RowHammer DIMMs compared to a baseline unmodified gem5 for NPB and GAPBS, respectively. To maintain the additional data structures to simulate RowHammer, HammerSim consumes 53.9% and 65% additional host memory for NPB and GAPBS, respectively.

Table 4 also shows the outcome of the experiments, *i.e.*, errors induced via RowHammer. While every workload had bitflips (column 6), not all workloads failed to reach the end of ROI. This means that there were silent errors that propagated to the end of the program. We did validate the results at the end of the ROI. We saw that workloads like mg and ft successfully validated their results, which means that RowHammer bitflips did not affect the outcome of the workload. This has been shown in prior research as well [15]. We plan on investigating this aspect of fault-tolerant workload behavior in the future.

Benchmark	Host Runtime (s)		Host Memory (Bytes)		Status	Notes
	base	ours	base	ours		
cg	19348.39	23165.54	1980816	3188016	✗	Seg. Fault
ep	162094.28	173420.81	2585996	3807536	✓	Silent errors
ft	70689.88	77007.16	2489744	3709232	✓	Silent errors
is	7786.01	9343.82	1952144	3166508	✗	Seg. Fault
lu	703200.95	703200.95	2794892	4004140	✓	Silent errors
mg	26627.67	30435.92	1975696	3624236	✓	Silent errors
sp	641277.22	755053.48	2785680	4000048	✓	Silent errors
bc	5277.73	5064.94	1878412	3091760	✓	Silent errors
bfs	254.57	413.62	1521036	2733360	✓	Silent errors
cc	399.36	566.8	1503632	2711856	✓	Silent errors
pr	6303.33	8070.86	1874316	3092784	✓	Silent errors
tc	156894.62	204465.25	2553228	3774764	✗	Seg. Fault

TABLE 4: This table shows the simulation statistics of baseline gem5 and HammerSim. We also show the data corruption status (verification of the program status) of benign applications.

Takeaway 5: While HammerSim is 14.7% slower than gem5 overall, it allows the study of soft errors on full-system applications.

5.4. Additional Results

HammerSim allows us to use statistically generated device maps. This generalizes the usability of the tool since we can use generic device maps based on some process variation-based probability distribution. Visualized results of the simulation on a specific row with 8192 columns can be seen in Figures 11(a) - 11(e). The hardware counterpart of the actual weak cell map is shown in Figure 4. The measured similarity index between the real and the simulated run is 0.31 in terms of JS Divergence [64] value.

6. Related Works

RowHammer is widely studied using trace-based simulators [86, 46, 97, 70, 7]. The major limitation of this approach is that it does not consider any of the OS effects or influences while studying both RowHammer attacks and defenses. DRAM DIMMs strictly operate based on timing. This led to the hypothesis that RowHammer mitigations or *inhibitor* sends ACT commands to possible victim rows during tREFI time, when the device is locked for normal reads and writes [20, 36, 30]. This further implies that adding victim ACTs whenever a RowHammer threshold is crossed may not be practical to implement by DRAM vendors.

(a) Run 1 (b) Run 2 (c) Run 3 (d) Run 4 (e) Superimposed Image

Figure 11: Simulated RowHammer result of bitflips on a single row. This set is generated using a variation map taken from an actual DRAM DIMM's row (1278) (Figure 4)

gem5 has been used to study architectural and micro-architectural vulnerabilities in the past [16, 51]. There have been attempts on implementing RowHammer at a system-level in the past using gem5 [17, 18, 89]. Most of these works use an external trace-based simulator like ramulator [44] or DRAMsim3 [48] as the backend memory. France *et al.* [17] use ramulator [44] to simulate the backend memory and bitflips. The authors also did a vulnerability assessment of the RowHammer attack [18] using a similar setup. Hammulator [89] uses DRAMsim3 [48] as the backend memory.

While the aforementioned approaches are similar to HammerSim, our technique adds the ability to experiment with different probability distributions. This allows further exploration of implications of RowHammer in terms of applications and experiment system-level RowHammer mitigations This allows researchers to perform system-level exploits and mitigations while taking different probability distributions into account.

Data corruption at a system-level allows researchers to deterministically perform RowHammer attacks and exploit system-sensitive data. This was demonstrated in rowhammer-test [83] where the authors gained access to the page table from userspace by correctly flipping the access bits. HammerSim improves upon these aforementioned works by adding uniqueness to the simulated DIMMs to represent real-world DIMMs via a device map. This not only limits HammerSim to be a RowHammer simulator, but also do DRAM/memory variation studies using simulators, making them closer to its hardware counterpart [22] at a system-level.

7. Conclusion

In this work, we introduced HammerSim, a system-level framework for modeling RowHammer within the gem5 simulator. HammerSim enables researchers to explore diverse RowHammer access patterns under realistic full-system conditions, including OS and cache interactions. By incorporating device maps, the tool captures per-DIMM vulnerability characteristics, allowing fine-grained analysis of bitflip patterns and their implications. Our framework supports both synthetic attack traces and real RowHammer programs, enabling comprehensive evaluation of mitigation strategies. While the mitigations demonstrated in this paper are intentionally simple, HammerSim provides an extensible foundation for prototyping and benchmarking advanced defenses. Looking ahead, we plan to enhance HammerSim with richer mitigation models and introduce a quantitative RowHammer vulnerability metric at the system level, fostering rigorous and reproducible research in DRAM reliability and security.

References

[1] B. Aichinger, "Ddr memory errors caused by row hammer," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 2015, pp. 1–5.

[2] Z. Al-Ars, "Dram fault analysis and test generation," 2005.

[3] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2872362.2872390

[4] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS Parallel Benchmarks," *The International Journal of Supercomputing Applic ations*, vol. 5, no. 3, pp. 63–73, 1991.

[5] K. S. Bains and J. B. Halbert, "Distributed row hammer tracking," 2012, US Patent US20140095780A1.

[6] S. Beamer, K. Asanović, and D. Patterson, "The GAP Benchmark Suite," *arXiv preprint arXiv:1508.03619*, 2015.

[7] T. Bennett, S. Saroiu, A. Wolman, L. Cojocar, Microsoft, and Avant-Gray, "Panopticon: A complete in-dram rowhammer mitigation," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235420813

[8] S. Bhattacharya and D. Mukhopadhyay, "Curious case of rowhammer: Flipping secret exponent bits using timing analysis," Cryptology ePrint Archive, Paper 2016/618, 2016. [Online]. Available: https://eprint.iacr.org/2016/618

[9] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[10] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup est machina: Memory deduplication as an advanced exploitation vector," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 987–1004.

[11] B. R. Bruce, A. Akram, H. Nguyen, K. Roarty, M. Samani, M. Friborz, T. Reddy, M. D. Sinclair, and J. Lowe-Power, "Enabling reproducible and agile full-system simulation," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 183–193.

[12] 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 *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 712–728.

[13] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 55–71.

[14] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized many-sided rowhammer attacks from JavaScript," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1001–1018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ridder

[15] F. de Ridder, P. Jattke, and K. Razavi, "Softhammer: Exploiting rowhammer bit flips without crashing," in *5th Workshop on DRAM Security (DRAMSec)*. ETH Zurich, 2025.

[16] Q. Forcioli, J.-L. Danger, and S. Chaudhuri, "A gem5 based platform for micro-architectural security analysis," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 91–99. [Online]. Available: https://doi.org/10.1145/3623652.3623674

[17] L. France, F. Bruguier, M. Mushtaq, D. Novo, and P. Benoit, "Implementation of rowhammer effect in gem5," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236634412

[18] L. France, M. Mushtaq, F. Bruguier, D. Novo, and P. Benoit, "Vulnerability assessment of the rowhammer attack using machine learning and the gem5 simulator - work in progress," *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233384386

[19] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: Accelerating microarchitectural attacks with the gpu," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 195–210.

[20] 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," 2020. [Online]. Available: https://arxiv.org/abs/2004.01807

[21] L. Gerlach, F. Thomas, R. Pietsch, and M. Schwarz, "A rowhammer reproduction study using the blacksmith fuzzer," in *ESORICS*, 2023.

[22] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, M. O'Connor, and O. Mutlu, "What your dram power models are not telling you: Lessons from a detailed experimental study," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, dec 2018. [Online]. Available: https://doi.org/10.1145/3224419

[23] Google LLC, ""Half-Double": Next-Row-Over Assisted Rowhammer," 2021. [Online]. Available: https://raw.githubusercontent.com/google/hammer-kit/main/20210525_half_double.pdf

[24] K. Goswami, D. S. Banerjee, and S. Das, "Towards enhanced system efficiency while mitigating row hammer," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3458749

[25] K. Goswami, S. Das, S. Satapathy, and D. S. Banerjee, "A case for amplifying row hammer attacks via cell-coupling in dram devices," in *Proceedings of the 2022 International Symposium on Memory Systems*, ser. MEMSYS '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3565053.3565056

[26] 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," 2018. [Online]. Available: https://arxiv.org/abs/1710.00551

[27] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 300–321.

[28] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. N. Udipi, "Simulating dram controllers for future system architecture exploration," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 201–210.

[29] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu, "Crow: A low-cost substrate for improving dram performance, energy efficiency, and reliability," in *Proceedings of the 46th international symposium on computer architecture*, 2019, pp. 129–142.

[30] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications," 2021. [Online]. Available: https://arxiv.org/abs/2110.10603

[31] 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 *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 241–252.

[32] Y. Hu, N. Brown, Y. Chen, J. Bakita, T. Chen, D. Genkin, and A. Kwong, "Gddrhammer: Greatly disturbing dram rows—cross-component rowhammer attacks from modern gpus," in *Proceedings of the 2026 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE*, 2026.

[33] Hynix Semiconductor, "Datasheet for 1Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR," Tech. Rep. H5GQ1H24AFR, 2009.

[34] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[35] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, ser. SysTEX'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3152701.3152709

[36] P. Jattke, V. Van Der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 716–734.

[37] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," 2022-05, https://github.com/comsec-group/blacksmith.

[38] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: Rowhammer attacks on AMD zen-based platforms," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1615–1633. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/jattke

[39] JEDEC, "DDR5 SDRAM," Tech. Rep. JESD79-5B, August 2022.

[40] Y. Jiang, H. Zhu, D. Sullivan, X. Guo, X. Zhang, and Y. Jin, "Quantifying rowhammer vulnerability for dram security," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 73–78.

[41] I. Kang, W. Wang, J. Kim, S. van Schaik, Y. Tobah, D. Genkin, A. Kwong, and Y. Yarom, "Sledgehammer: Amplifying rowhammer via bank-level parallelism."

[42] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 638–651.

[43] 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 *Proceeding of the 41st Annual International Symposium on Computer Architecture*. IEEE Press, 2014.

[44] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[45] 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 *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3807–3824. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double

[46] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 385–396. [Online]. Available: https://doi.org/10.1145/3307650.3322232

[47] D. Li, D. Liu, Y. Ren, Z. Wang, Y. Sun, Z. Guan, Q. Wu, and J. Liu, "Fphammer: A device identification framework based on dram fingerprinting," in *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2023, pp. 1031–1040.

[48] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[49] C. S. Lin, J. Qu, and G. Saileshwar, "GPUHammer: Rowhammer attacks on GPU memories are practical," in *34th USENIX Security Symposium (USENIX Security 25)*. Seattle, WA: USENIX Association, Aug. 2025, pp. 5719–5738. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/lin-shaopeng

[50] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "Moesi-prime: Preventing coherence-induced hammering in commodity workloads," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 670–684. [Online]. Available: https://doi.org/10.1145/3470496.3527427

[51] J. Lowe-Power, "Visualizing spectre with gem5," June 2018. [Online]. Available: http://www.lowepower.com/jason/visualizing-spectre-with-gem5.html

[52] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[53] 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 *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–18.

[54] E. Manzhosov and S. Sethumadhavan, "Polymorphic error correction," in *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '24. IEEE Press, 2024, p. 246–262. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00027

[55] M. Marazzi and K. Razavi, "Risc-h: Rowhammer attacks on risc-v," in *4th Workshop on DRAM Security (DRAMSec) co-located with ISCA 2024*. ETH Zurich, 2024.

[56] D. Meyer, P. Jattke, M. Marazzi, S. Qazi, D. Moghimi, and K. Razavi, "Phoenix: Rowhammer attacks on ddr5 with self-correcting synchronization," 2026.

[57] Micron Technology, "TN-46-12: Mobile DRAM Power-Saving Features and Power Calculations," Tech. Rep. TN46_12.

[58] ——, "DDR2 SDRAM ," Tech. Rep. MT47H512M4,MT47H256M8,MT47H128M16, 2006.

[59] ——, "TN-41-01: Calculating Memory System Power for DDR3," Tech. Rep. TN41_01DDR3, January 2007.

[60] ——, "1.35V DDR3L SDRAM SODIMM," Tech. Rep. MT16KTF51264HZ, MT16KTF1G64HZ, 2011.

[61] Micron Technology,, "DDR4 SDRAM," Tech. Rep. MT40A2G4, MT40A1G8, MT40A512M16, 2015.

[62] Micron Technology, "TN-ED-03: GDDR6: The Next-Generation Graphics DRAM ," Tech. Rep. TN-ED-03: GDDR6, 2017.

[63] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 39, no. 8, p. 1555–1571, aug 2020. [Online]. Available: https://doi.org/10.1109/TCAD.2019.2915318

[64] Notes on AI, "Jensen-Shannon Divergence," https://notesonai.com/Jensen%E2%80%93Shannon+Divergence.

[65] A. Olgun, H. Hassan, A. G. Yaglikci, Y. C. Tugrul, 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," *IEEE TCAD*, 2023.

[66] A. Olgun, Y. C. Tugrul, N. Bostanci, I. E. Yuksel, H. Luo, S. Rhyner, A. G. Yaglikci, G. F. Oliveira, and O. Mutlu, "Abacus: All-bank activation counters for scalable and low overhead rowhammer mitigation," 2023.

[67] 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," *arXiv preprint arXiv:2210.04084*, 2022.

[68] L. Orosa, A. G. Yaglikci, 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-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1182–1197. [Online]. Available: https://doi.org/10.1145/3466752.3480069

[69] K. Park, D. Yun, and S. Baeg, "Statistical distributions of row-hammering induced failures in ddr3 components," *Microelectronics Reliability*, vol. 67, pp. 143–149, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271416304000

[70] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. Ho Ahn, and J. W. Lee, "Graphene: Strong yet lightweight row hammer protection," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1–13.

[71] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.

[72] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for Cross-CPU attacks," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 565–581. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[73] A. Plin, F. Fauberteau, and N. Nguyen, "Opengl gpu-based rowhammer attack (work in progress)," in *European Symposium on Research in Computer Security*. Springer, 2025, pp. 347–358.

[74] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*, 2012, pp. 392–397.

[75] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, 2014.

[76] M. Qureshi, "Rethinking ecc in the era of row-hammer," *DRAMSec*, 2021.

[77] V. Ramadas, D. Kouchekinia, N. Osuji, and M. D. Sinclair, "Closing the gap: Improving the accuracy of gem5's gpu models." 5th gem5 Users' Workshop, 2023.

[78] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[79] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized row-swap: mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1056–1069. [Online]. Available: https://doi.org/10.1145/3503222.3507716

[80] Samsung Electronics, "DDR4 SDRAM," Tech. Rep., 2014. [Online]. Available: https://www.samsung.com/semiconductor/global.semi/file/resource/2017/11/DDR4_Device_Operations_Rev11_Oct_14-0.pdf

[81] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[82] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, "Intrinsic rowhammer PUFs: Leveraging the rowhammer effect for improved security," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, may 2017. [Online]. Available: https://doi.org/10.1109%2Fhst.2017.7951729

[83] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[84] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 18–21, 2017. [Online]. Available: https://doi.org/10.1109/LCA.2016.2614497

[85] S. M. Seyedzadeh, D. Kline, A. K. Jones, and R. Melhem, "Mitigating bitline crosstalk noise in dram memories," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 205–216. [Online]. Available: https://doi.org/10.1145/3132402.3132410

[86] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3061639.3062281

[87] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating software mitigations against rowhammer: A surgical precision hammer," in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, pp. 47–66.

[88] M. Technology, "Ddr4 rdimm 64gb x72 ecc module datasheet," 2022, accessed: 2025-11-29. [Online]. Available: https://advdownload.advantech.com/productfile/PIS/96D4-64G3200ER-M2/file/96D4-64G3200ER-M2_datasheet20220926100802.pdf

[89] F. Thomas, L. Gerlach, and M. Schwarz, "Hammulator: Simulate now-exploit later," in *3rd Workshop on DRAM Security*, 2023.

[90] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "Spechammer: Combining spectre and rowhammer for new speculative attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 681–698.

[91] 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2976749.2978406

[92] H. Venugopalan, K. Goswami, Z. A. Din, J. Lowe-Power, S. T. King, and Z. Shafiq, "Centauri: Practical rowhammer fingerprinting," *arXiv preprint arXiv:2307.00143*, 2023.

[93] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: A Memory System Simulator," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 100–107, November 2005.

[94] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 374–389.

[95] S. C. Woo, W. Elsasser, M. Hamburg, E. Linstadt, M. R. Miller, T. Song, and J. Tringali, "Rampart: Rowhammer mitigation and repair for server memory systems," 2023.

[96] X.-C. Wu, T. Sherwood, F. T. Chong, and Y. Li, "Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 645–657. [Online]. Available: https://doi.org/10.1145/3297858.3304039

[97] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 345–358. [Online]. Available: https://doi.org/10.1109/HPCA51647.2021.00037

[98] S. Zeitouni, D. Gens, and A.-R. Sadeghi, "It's hammer time: How to attack (rowhammer-based) dram-pufs," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196065

[99] B. Zhao, Y. Du, J. Yang, and Y. Zhang, "Process Variation-Aware Nonuniform Cache Management in a 3D Die-Stacked Multicore Processor," vol. 62, no. 11, Nov 2013, pp. 2252–2265.

Appendix

1. Abstract

HammerSim is a full-system RowHammer modeling framework built as a fork of the gem5 architectural simulator. It integrates probability-driven, hardware-validated bitflip modeling directly into gem5's DRAM memory interface, enabling researchers to study RowHammer attacks, hardware/software mitigations, and functional SECDED ECC correction, all under an unmodified guest OS and real application workloads. This artifact provides the complete HammerSim source (a fork of gem5), example synthetic-traffic and full-system configuration scripts, and a hardware-derived DRAM device (weak-cell) map, which together were used to produce the RowHammer characterization, mitigation, and data-corruption results reported in the paper. This submission targets the **Available** badge: the artifact is publicly hosted, archived with a persistent identifier, and documented sufficiently to support reproducing the paper's setup.

2. Artifact check-list (meta-information)

- **Algorithm:** Probability-driven RowHammer bitflip injection (per-row activation counters with weak-cell, single/double-sided, and half-double probability distributions); reverse-engineered Target Row Refresh (TRR) mitigation variants and PARA; functional SECDED ECC (1-bit correct / 2-bit detect).
- **Program:** HammerSim (C++/Python), a fork of the gem5 architectural simulator.
- **Compilation:** SCons, g++ or clang (standard gem5 build toolchain).
- **Transformations:** N/A.
- **Binary:** Not distributed; built from source as `gem5.opt`.
- **Model:** DDR3/DDR4 DRAM timing and RowHammer vulnerability models integrated into gem5's DRAM interface.
- **Data set:** Hardware-derived DRAM weak-cell/device map (`prob-005.json.zip`, statistically generated with VARIUS from real DDR4 DIMM characterization); example RowHammer access-pattern traces under `configs/dram/rowhammer/TrafficGen`.
- **Run-time environment:** Linux x86_64, Python 3.
- **Hardware:** Any x86_64 host capable of compiling gem5. KVM-capable hardware is only needed for the accelerated full-system configurations under `FSConfigs`; not required to build or run the traffic-generator configurations.
- **Run-time state:** N/A.
- **Execution:** Command-line invocation of `gem5.opt` against a Python configuration script (traffic-generator or full-system).
- **Metrics:** RowHammer bitflip counts, TRR mitigation statistics, ECC correction/detection statistics, simulation wall-clock time and host memory overhead.
- **Output:** gem5 `stats.txt`; stdout bitflip/data-corruption logs; optional TRR/RowHammer stat dumps (`trr_stat_dump`, `rh_stat_dump`).
- **Experiments:** RowHammer hardware-validation case study (JS-divergence comparison against real DDR4 DIMMs); full-system data corruption and ECC case study using Google's `rowhammer-test`; benign-workload susceptibility study using NAS Parallel Benchmarks and GAPBS.
- **How much disk space required (approximately)?:** 10 GiB for a gem5 build; 100 MiB for the included device map (unzipped); several additional GiB if the optional full-system kernel/disk images are downloaded.
- **How much time is needed to prepare workflow (approximately)?:** 30–60 minutes (clone, install build dependencies, compile gem5.opt).
- **How much time is needed to complete experiments (approximately)?:** Traffic-generator experiments run in minutes and full-system experiments (`rowhammer-test`) take roughly an hour and require a separately built kernel/disk image. Benign applications from NPB and GAPBS can take up to two weeks to finish the workload end-to-end.
- **Publicly available?:** Yes: https://github.com/darchr/gem5-rowhammer.
- **Code licenses (if publicly available)?:** BSD 3-Clause "New" or "Revised" License (inherited from the upstream gem5 project this repository extends).
- **Data licenses (if publicly available)?:** No third-party restrictions; the device map is generated by the authors.
- **Workflow automation framework used?:** None – manual SCons build and direct `gem5.opt` invocation of Python config scripts.
- **Archived (provide DOI)?:** 10.5281/zenodo.21843464

3. Description

3.1. How to access. The source is hosted on Github (a public fork of gem5). For artifact evaluation, use the tagged release `iiswc` (branch `artifact-iiswc`), also archived at 10.5281/zenodo.21843464 for long-term access.

3.2. Hardware dependencies. None beyond a standard x86_64 development machine. KVM acceleration (an Intel VT-x or AMD-V capable CPU) is only needed for the optional full-system configurations that boot a guest Linux kernel.

3.3. Software dependencies. A C++ compiler (g++ or clang), SCons, Python 3, zlib, m4, and (optionally) protobuf, per gem5's standard build requirements (https://www.gem5.org/documentation/general_docs/building); the

Category	Parameter	Description
RowHammer simulation parameters	device_file	Path to a variation map.
	rowhammer_threshold	Standard RowHammer bitflip threshold.
	single_sided_prob	Probability of observing a single-sided RowHammer bitflip.
	double_sided_prob	Probability of observing a double-sided RowHammer bitflip.
	half_double_prob	Probability of observing a half-double RowHammer bitflip.
RowHammer mitigation parameters	trr_variant	Mitigation variant to use during simulation.
	trr_threshold	Every mitigation needs to have a threshold to send refreshes.
	companion_threshold	Some mitigations use multiple tables. This structure addresses that.
	counter_table_length	The length of the mitigation table.
	companion_table_length	The length of the secondary counter table.
Features	enable_memory_corruption	Simulate live memory corruption.
ECC parameters	enable_ecc	Binary input to enable ECC.
	p_matrix	A plain text file of a p_matrix.
	ecc_algorithm	To switch between multiple ECC algorighms.
Utilities	trr_stats_dump	Dumps all TRR variable counts.
	rh_stat_file	Dumps all bitflip variable counts.
	synthetic_traffic	To enable synthetic traffic.

TABLE 5: All parameters available to the user to simulate RowHammer in HammerSim.

bundled `ext/json` dependency (build instructions in `ext/json/README`).

3.4. Data sets. A pre-generated DRAM weak-cell/device map, `prob-005.json.zip` (located in util/hammersim/), statistically generated with VARIUS from real DDR4 DIMM characterization data (see paper Section 3.2). More device maps from 10 different DIMMs from the same DRAM vendor are also available in the same directory. The format of the device map is:

```
{
    "rank_number": {
        "bank_number": {
            "row_number": [..,
  list_of_all_weak_columns, .. ],
        }
    }
}

```

Listing 1: Device Map Format

Additional resources including the disk image and the kernel can be compiled via:

```
git clone https://github.com/kaustav-goswami/
    gem5-resources.git
cd gem5-resources
git checkout rowhammer
cd src/rowhammer-fs
./build-x86.sh 22.04
cd ~
wget https://www.kernel.org/pub/linux/kernel/v5.
    x/linux-5.4.49.tar.xz
tar xvf linux-5.4.49.tar.xz
cd linux-5.4.49.tar.xz
menu config  # Use the default build
make -j32
```

Listing 2: Creating the disk image and compiling the kernel

3.5. Models. N/A beyond the DDR3/DDR4 DRAM timing models already present in gem5's DRAM interface, extended with RowHammer/TRR/ECC logic.

4. Installation

```
git clone https://github.com/darchr/gem5-
    rowhammer.git
cd gem5-rowhammer/ext/json
mkdir build
cd build
cmake ..
make -j32
cd ../..
scons build/ALL/gem5.opt -j32
cd util/hammersim
unzip prob-005.json.zip
```

Listing 3: Compiling HammerSim

5. Experiment workflow

Two categories of ready-to-use configuration scripts are provided under `configs/dram/rowhammer/`: (a) `TrafficGen/`: synthetic gem5 traffic-generator scripts that replay vendor-specific RowHammer access patterns against a modeled DIMM without booting an OS; and (b) `FSConfigs/`: full-system scripts that boot a Linux guest and run RowHammer workloads (e.g. Google's `rowhammer-test`) with live data corruption and optional ECC enabled. Both are invoked as standard gem5 Python configuration scripts:

```
build/ALL/gem5.opt \
  --outdir=sample-rowhammer-test \
  <gem5/script/in/configs/dram/rowhammer>
```

Listing 4: Running HammerSim

6. Evaluation and expected results

This submission targets the "Available" badge only. The repository contains the code, configuration scripts, and dataset used to produce the paper's case studies: (a) hardware-validation via JS divergence (Section 5.1), (b) full-system data corruption and ECC correction with `rowhammer-test` (Section 5.2), and (c) benign-workload susceptibility with NAS Parallel Benchmarks / GAPBS (Section 5.3). We expect a reviewer can confirm the artifact is complete and could support reproducing those results. A successful traffic-generator run produces a bitflip count on stdout and in gem5's `stats.txt`, consistent in form with paper Figure 9.

7. Experiment customization

All the parameters introduces by HammerSim to simulate RowHammer are listed in Table 5.

8. Notes

HammerSim is an actively developed research fork of gem5. We expect the repository to be updated frequently.

9. Methodology

We are only targeting the "Available" badge.