Loaded Dice: Solving the Non-Selection Problem for Scalable Probabilistic RowHammer Defense

Jeonghyun Woo*
University of British Columbia
jhwoo36@ece.ubc.ca

Junsu Kim*
University of British Columbia
junsukim@ece.ubc.ca

Aamer Jaleel
NVIDIA
ajaleel@nvidia.com

Prashant J. Nair
University of British Columbia
prashantnair@ece.ubc.ca

***Abstract*—DRAM scaling has exacerbated the RowHammer vulnerability. To counter this, JEDEC recently introduced Per Row Activation Counting (PRAC) with the Alert Back-Off protocol as an optional DDR5 feature. While promising, PRAC requires per-row counter cells that incur area overhead, and updating them on every activation lengthens DRAM timing parameters, degrading performance. Probabilistic mitigations such as MINT offer a lower-cost alternative by randomly selecting and mitigating rows within periodic mitigation windows. MINT is effective at higher thresholds (≥ 1000), but at lower thresholds, it must raise its mitigation rate to overcome the non-selection problem, where heavily hammered rows can repeatedly escape sampling. This fixed-rate scaling reduces effective memory bandwidth even when no attack is present.**

To overcome this limitation, we propose PrISM, an intersection-based probabilistic mitigation that correlates sampled rows across windows using a Sampled History Queue (SHQ). PrISM samples a few activation slots per window, stores sampled-but-unmitigated rows in the SHQ, and requests an additional mitigation through the existing Alert Back-Off protocol when a sampled row reappears in this history. This allows PrISM to increase mitigation only when persistent row activity is observed, without globally increasing the fixed mitigation rate. At the threshold of 500, PrISM incurs a negligible 0.2% average slowdown compared to 14% for PRAC, with no DRAM array changes or per-row counters and only 625B of SRAM per bank, one to two orders of magnitude less than prior secure counter-based in-DRAM defenses. Compared to MINT, PrISM provides better scalability at low thresholds, reducing average slowdown from 10.7% to 1.5% at a threshold of 250, a 7.1× reduction. PrISM is open-sourced at https://github.com/STAR-Laboratory/prism.

I. Introduction

DRAM technology scaling has enabled large main memory capacities for data-intensive workloads. At the same time, smaller cells and tighter noise margins have made DRAM more vulnerable to disturbance effects. The most prominent example is the RowHammer vulnerability: a read-disturb phenomenon where repeatedly activating a DRAM row induces bit flips in physically adjacent rows [40], [48]. Over the past decade, RowHammer has become a serious security problem, with several attacks compromising real systems [12], [16], [51], [54]. Meanwhile, aggressive scaling has sharply reduced the RowHammer threshold (T_{RH}), the minimum number of activations needed to induce bit flips. In this work, we focus on the double-sided RowHammer threshold ($T_{RH\text{-}D}$), where two aggressor rows adjacent to a victim are alternately hammered.

*Both authors contributed equally to this work.

To counter RowHammer, JEDEC standardized Per Row Activation Counting (PRAC) for DDR5 [36]. PRAC maintains a per-row activation counter and uses the *Alert Back-Off (ABO)* protocol. When a counter reaches the Back-Off threshold, DRAM asserts Alert and the controller issues a Refresh Management (RFM) command while pausing regular requests for a fixed interval (e.g., 350ns). Recent designs such as QPRAC [101] and MOAT [71] show that PRAC can provide strong protection even at sub-100 T_{RH}, but at high cost [10], [98]. Because PRAC updates a per-row counter on every activation through a read-modify-write operation, it increases the tRP and tRC timing parameters, resulting in significant performance degradation. As Figure 1 shows, PRAC, implemented using QPRAC, incurs 14% average slowdown across all workloads and 21.8% slowdown on high-memory-intensity workloads ($\geq$10 row-buffer misses per kilo-instruction), largely independent of $T_{RH\text{-}D}$ (see Section V for methodology). PRAC also incurs notable area overhead due to its per-row counters [27], [72]. Since PRAC is an *optional* DDR5 feature [36] and its commercial adoption is uncertain given these costs, there is a clear need for practical low-overhead mitigations for near-term DDR5 systems [93].

Probabilistic in-DRAM mitigations offer a much lower-cost alternative to PRAC [30], [31], [72], [106]. These schemes avoid per-row counters by randomly mitigating a subset of recently activated rows using either Target Row Refresh (TRR), which borrows time from periodic refreshes [25], [35], or RFM, which temporarily blocks normal memory service while

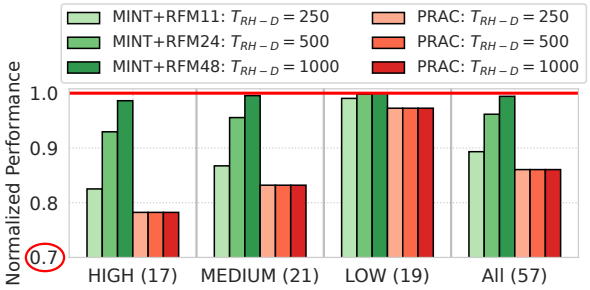

Fig. 1. Performance overhead of PRAC [101] and MINT [72] under varying double-sided RowHammer thresholds ($T_{RH\text{-}D}$). On high-memory-intensity workloads, PRAC incurs an average 21.8% slowdown due to updates to the activation counter on every activation. In contrast, MINT incurs only 1.4% overhead at $T_{RH\text{-}D}$ of 1000, but its slowdown increases to 17.5% at $T_{RH\text{-}D}$ of 250 as lower thresholds require more frequent mitigations.

DRAM performs the required mitigation. Unless otherwise stated, we assume one TRR mitigation opportunity every two tREFI intervals. Because each RFM consumes memory bandwidth and delays regular requests, probabilistic schemes are most effective when RFMs are issued only occasionally. This works well at higher thresholds ($T_{RH\text{-}D} \geq 1000$), where infrequent mitigations are enough to provide security. For example, MINT [72] requires mitigations only every 48 activations at $T_{RH\text{-}D}$ of 1000, resulting in only a 1.4% slowdown on high-memory-intensity workloads. This makes MINT highly attractive at sufficiently high thresholds, especially given its extremely small in-DRAM storage cost.

At lower thresholds, however, probabilistic schemes face a statistical barrier: the **non-selection problem**. Each mitigation window mitigates at most one randomly chosen row, so a heavily hammered aggressor may be skipped across many windows and remain unmitigated long enough to induce bit flips. To maintain security, MINT *statically* increases its mitigation rate, issuing RFMs more frequently even when no aggressor is present. This raises overhead as $T_{RH\text{-}D}$ drops, especially for high-memory-intensity workloads. As Figure 1 shows, MINT performs mitigations every 24 activations at $T_{RH\text{-}D}$ of 500, increasing slowdown to 7.1%, and every 11 activations at $T_{RH\text{-}D}$ of 250, increasing slowdown to 17.5%. These results motivate a probabilistic defense that scales to lower thresholds without uniformly increasing mitigation frequency.

To this end, we propose **PrISM**, Probabilistic Intersection-based Sampling Mitigation. The key insight is that rows that are repeatedly activated are more likely to reappear across mitigation windows, whereas benign rows rarely reappear in the sampled history. PrISM exploits this temporal correlation to address the *non-selection problem*. Instead of statically raising the mitigation rate, PrISM samples a small set of activation slots in each window and tracks their row addresses across a recent *lookback window*. When a newly sampled row matches a row already in this history, PrISM requests an *additional* mitigation through the existing ABO protocol.

This mechanism allows PrISM to increase mitigation only when persistent row activity is observed, without scaling the default mitigation rate. In MINT, a mitigation window with W activation slots selects only one row, so a persistent aggressor is selected with probability $1/W$, about 1.4% when $W = 72$. PrISM instead samples R slots per window and remembers sampled-but-unmitigated rows for L windows using the Sampled History Queue (SHQ). Thus, a row that appears in every window has probability R/W of being sampled in each window, and its chance of appearing at least once in the lookback history is roughly $1 - (1 - R/W)^L$. With $W = 72$, $R = 7$, and $L = 41$, this probability exceeds 98%, making a persistent aggressor highly likely to create an SHQ intersection while keeping the default mitigation rate low.

Crucially, PrISM requests few additional Alert-induced RFMs for benign applications. A benign row accessed only occasionally is unlikely to be sampled repeatedly within the SHQ lookback window, so it seldom creates intersections and therefore rarely triggers additional mitigations. Thus, PrISM preserves the counter-free nature of probabilistic mitigations while requesting additional mitigations only for repeated sampled activity. In contrast, MINT maintains security at low $T_{RH\text{-}D}$ by increasing the mitigation rate for all workloads, reducing effective memory bandwidth.

PrISM is fully compatible with the existing JEDEC ABO protocol and requires no DRAM array or interface changes. Compared to PRAC, PrISM avoids per-row counters and counter updates on every activation, which lengthen DRAM timing parameters and cause 14% average slowdown. Compared to MINT, PrISM improves scalability at low $T_{RH\text{-}D}$ by keeping the default mitigation rate low and requesting additional mitigations only when SHQ intersections indicate repeated sampled activity. At $T_{RH\text{-}D}$ of 500, PrISM achieves a negligible 0.2% average slowdown while requiring only 625B of SRAM per bank, which is about 20× and 170× smaller than prior secure counter-based in-DRAM mitigations such as Mithril [46] and ProTRR [58], respectively. Even at an ultra-low $T_{RH\text{-}D}$ of 250, where MINT incurs 10.7% average slowdown due to frequent fixed-rate mitigations, PrISM incurs only 1.5% average slowdown, a 7.1× reduction.

Summary of Contributions:

- We identify the **non-selection problem** as the key barrier for fixed-rate probabilistic RowHammer defenses at low $T_{RH\text{-}D}$, and propose **PrISM**, which addresses it by correlating sampled row addresses across windows.
- PrISM uses SHQ intersections to request additional mitigations only for repeated sampled activity, keeping the default RFM rate low. It reuses the existing ABO protocol and requires no changes to the DRAM array or interface.
- PrISM avoids PRAC's costly per-row counters and counter updates on each activation. At $T_{RH\text{-}D}$ of 500, PrISM achieves a negligible 0.2% average slowdown, compared to 14% for PRAC, while requiring only 625B of per-bank SRAM.
- We show that PrISM improves low-threshold scalability over MINT by avoiding the need to uniformly increase RFM frequency. On high-memory-intensity workloads, PrISM reduces slowdown from 7.1% to 0.5% at $T_{RH\text{-}D}$ of 500 and from 17.5% to 2.5% at $T_{RH\text{-}D}$ of 250.

II. Background and Motivation

A. Threat Model

We consider a DRAM-based system vulnerable to RowHammer. The adversary is unprivileged, knows the deployed in-DRAM defenses, and can craft tailored access patterns to bypass them [3], [33], [60]. In our evaluation, all probabilistic mitigations, including MINT and PrISM, use fractal mitigation [70] to defend against transitive attacks [50].

We focus on activation-driven RowHammer. RowPress [56] and ColumnDisturb [114] are outside our primary threat model; Appendix C discusses possible extensions.

B. The RowHammer Vulnerability

RowHammer is a read-disturbance phenomenon in which repeated activations of aggressor rows disturb charge in nearby

victim rows and can induce bit flips [45], [48], [83], [85], [96]. The RowHammer threshold (T_{RH}), the minimum number of activations needed to induce a bit flip, has decreased significantly across DRAM generations [42]. We use the double-sided threshold ($T_{RH\text{-}D}$), where two aggressors adjacent to a victim are alternately hammered, as our primary metric.

RowHammer is both a security and a reliability concern. It has enabled attacks across CPUs and GPUs [28], [53], [97], including privilege escalation [54], [86], ML model degradation [26], [109], and other exploits [12], [13], [21], [22], [32], [74], [75]. Recent work also shows that RowHammer mitigations can introduce denial-of-service and timing-channel attack surfaces [7], [102], [104]. Bit flips have even been observed under benign workloads [55], further underscoring the need for secure and low-overhead RowHammer defenses.

C. Target Row Refresh and Refresh Management

Commodity DRAM has relied primarily on two in-DRAM RowHammer mitigation methods:

Target Row Refresh (TRR): TRR tracks potential aggressor rows using small per-bank trackers, typically with 4–28 entries, or probabilistic sampling [25], [33], [37]. It then refreshes nearby victim rows within a fixed blast radius (e.g., BR = 2) during periodic refreshes by borrowing time from the refresh cycle (tRFC)[1] [25], [66]. However, TRR's limited tracking capacity makes it vulnerable to crafted access patterns, and prior works have bypassed TRR on DDR4 [19], [33], [50] and DDR5 devices [34], [60].

Refresh Management (RFM): To address TRR's limitations, DDR5 introduced RFM [36]. The memory controller monitors per-bank activations and issues an RFM once a threshold is reached, allowing DRAM to perform RowHammer mitigations. DDR5 defines two RFM types: All-Bank RFM (RFM_{ab}), which applies mitigations across all banks, and Same-Bank RFM (RFM_{sb}), which applies mitigations to the same bank ID across all bank groups (e.g., Bank 0 in every bank group).

D. Per Row Activation Counting (PRAC)

To address the worsening RowHammer vulnerability, JEDEC standardized *Per Row Activation Counting (PRAC)* in the DDR5 specification [36]. PRAC enables precise aggressor tracking through two primary mechanisms.

Per-Row Activation Counters: Each DRAM row is equipped with counter cells [6]. On every activation, the corresponding counter is incremented via a read–modify–write during precharge, increasing core timings such as tRP and tRC.

Alert Back-Off (ABO) Protocol: PRAC also defines the Alert Back-Off (ABO) Protocol, which lets DRAM request mitigation time from the memory controller. When a counter or mitigation queue reaches the Back-Off threshold (N_{BO}), DRAM asserts Alert; after up to 180ns during which the controller may continue issuing regular DRAM commands, the controller issues a configured number of RFM_{ab} commands ($N_{mit} \in \{1, 2, 4\}$). Each RFM_{ab} stalls the entire channel for 350 ns, giving DRAM time to perform targeted refreshes. The current specification also requires an ABO_{Delay} of N_{mit} activations before a subsequent Alert.

Recent studies, including QPRAC [101] and MOAT [71], propose secure PRAC implementations that demonstrate robust RowHammer protection even at sub-100 T_{RH} [10]. In this paper, we use QPRAC as our baseline PRAC design.

E. Pitfall of PRAC: Performance and Area Overhead

Secure PRAC designs can provide strong protection even at sub-100 T_{RH}, but they suffer from two key drawbacks.

Performance Overhead: PRAC updates a per-row activation counter on every activation through a read–modify–write operation during precharge [36]. This increases key timing parameters: tRP rises from 16ns to 36ns (2.25×), and tRC rises from 48ns to 52ns [98]. As a result, row-buffer-conflict latency increases by 42%, while the longer tRC reduces the per-bank activation rate by 8%. Prior work reports an average slowdown of ∼10% due to PRAC timing overheads [10], [98].

We further show that PRAC's performance overhead grows sharply with interface speed. Figure 2 shows PRAC performance across DDR5 data rates. PRAC incurs a 2.2% average slowdown at 3200 MT/s, but this rises to 14% at 8000 MT/s, where shorter tRRD and tFAW allow the controller to issue activations across banks more rapidly, exposing PRAC's fixed timing penalties more often[2]. As DRAM data rates continue to scale (e.g., up to 17.6Gb/s in DDR6 [93]), these timing penalties may become more significant.

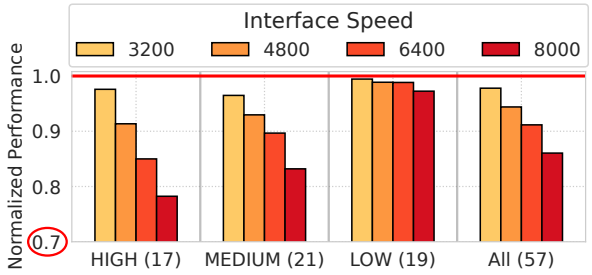

Fig. 2. Normalized performance of PRAC [101] across DDR5 interface speeds. At 3200 MT/s, PRAC incurs a 2.2% slowdown, which rises to 14% at 8000 MT/s due to amplified timing penalties at higher data rates.

Area Overhead: PRAC also incurs notable area overhead because each row requires additional counter cells and update logic. An early Samsung DSAC work [27] estimates that these additions increase overall DRAM core area by roughly 9%, which is significant for the price-sensitive DRAM market.

PRAC is currently defined as an *optional* feature in the DDR5 specification [36]. DDR5 is expected to remain the dominant DRAM technology for several years before DDR6, where PRAC is expected to see broader adoption and may become mandatory, as discussed at DRAMSec'24 [93]. Developing practical, low-overhead RowHammer mitigations for current and near-term DDR5 systems is therefore critical.

[1]We assume one TRR opportunity per two tREFI intervals, based on prior studies [25], [37]. Section VI-B evaluates sensitivity to this rate.

[2]Appendix A provides a detailed comparison with recent real-system PRAC performance measurements [43].

F. Scalability Limits of Probabilistic Mitigations

Prior work has proposed lightweight probabilistic defenses such as PrIDE [31] and MINT [72], which require only a small in-DRAM state. MINT randomly selects one activation slot per *mitigation window*, defined as the maximum number of activations between mitigations, and mitigates the corresponding row at the window's end. At $T_{RH\text{-}D} \geq 1000$, this design provides strong protection with negligible slowdown.

At lower $T_{RH\text{-}D}$, however, probabilistic defenses face a fundamental *non-selection problem*: random sampling can miss an aggressively hammered row across multiple windows, allowing it to remain unmitigated long enough to induce bit flips. MINT maintains security at lower thresholds by statically increasing its mitigation rate and issuing RFMs more frequently, even when aggressive behavior is absent. Each RFM temporarily blocks memory requests during mitigation, reducing effective DRAM bandwidth. For example, MINT issues an RFM every 24 activations at $T_{RH\text{-}D}$ of 500 and every 11 activations at $T_{RH\text{-}D}$ of 250, reducing available bandwidth by nearly 23% and 40%. On high-memory-intensity workloads, this causes 7.1% and 17.5% slowdown, respectively. Thus, while MINT remains highly storage-efficient, its fixed-rate scaling becomes increasingly costly at low $T_{RH\text{-}D}$. Our goal is to design a scalable probabilistic mitigation that preserves high performance even at low thresholds.

III. PrISM: A Scalable Probabilistic Mitigation

We propose PrISM, Probabilistic Intersection-based Sampling Mitigation, a scalable probabilistic RowHammer mitigation that solves the *non-selection problem*. Fixed-rate probabilistic defenses such as MINT [72] provide strong protection and require extremely small in-DRAM state, but at low thresholds ($T_{RH\text{-}D} \leq 500$), they must issue RFMs frequently even when the system is not under attack, reducing effective memory bandwidth and degrading performance.

To solve this, PrISM correlates sampled row addresses across mitigation windows and requests *additional* mitigations through the Alert Back-Off (ABO) protocol only when repeated sampled activity is observed. In each mitigation window, PrISM samples multiple activation slots and compares the sampled rows against a bounded history of previously sampled rows. We define an *intersection* as a match between a newly sampled row and a row in this history. A heavily activated row is more likely to reappear across windows and create intersections, making it eligible for additional mitigations. In contrast, benign rows accessed irregularly are less likely to create intersections, avoiding unnecessary mitigations.

This allows PrISM to use a larger default window W than MINT, reducing the default RFM rate while preserving strong RowHammer protection. As a result, PrISM scales to lower $T_{RH\text{-}D}$ without uniformly increasing mitigation frequency, preserving performance relative to fixed-rate probabilistic defenses. Compared to PRAC, PrISM avoids DRAM array changes, per-row counters, and counter updates on every activation, reducing both performance and area overhead.

A. Design of PrISM

PrISM enables intersection-based additional mitigations using three per-bank structures, as shown in Figure 3. The **Sampled Slot Queue (SSQ)** buffers row addresses of sampled and activated slots, from which PrISM selects the default mitigation candidate at the end of each window. The **Sampled History Queue (SHQ)** retains sampled-but-unselected rows from the previous L windows, called the lookback window, and serves as the history for intersection checks. When a newly sampled row matches an SHQ entry, PrISM enqueues the intersecting row into the **Pending Mitigation Queue (PMQ)** for additional mitigation. PrISM also enqueues one randomly selected non-intersecting sample as the default mitigation candidate, so each window still contributes one default probabilistic mitigation. The remaining sampled-but-unselected rows are inserted into the SHQ for future intersection checks.

The PMQ buffers rows selected for mitigation but not yet serviced. Each entry contains a row address and an activation counter that tracks activations while the row resides in the PMQ. At each mitigation opportunity, PrISM mitigates the highest-count entry and dequeues it; if the PMQ becomes full or any entry's counter exceeds the *tardiness threshold* (T_{PMQ}), PrISM requests additional mitigation through the existing ABO protocol. Unless otherwise stated, we use a 16-entry PMQ and set T_{PMQ} to 4.

By decoupling mitigation selection from mitigation service, the PMQ provides three benefits. First, it makes PrISM compatible with refresh and RFM postponement [36], since selected rows can wait until a mitigation opportunity becomes available. Second, because Alert is observed at the channel level while banks complete windows at different times, per-bank PMQs let other banks retain useful pending entries when any bank raises Alert; an ABO-triggered RFM can then service pending rows across banks, reducing future Alerts.

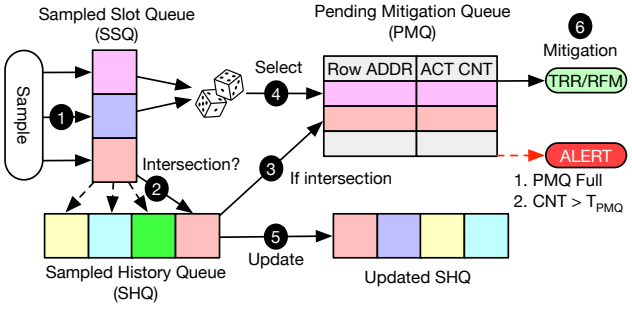

Fig. 3. Design and operation of PrISM. In each mitigation window, PrISM samples a few activation slots into the Sampled Slot Queue (SSQ) and checks them against the Sampled History Queue (SHQ), which stores sampled-but-unselected rows from recent windows. Rows that intersect with the SHQ are enqueued into the Pending Mitigation Queue (PMQ) for additional mitigation. PrISM also randomly selects one non-intersecting sampled row for the PMQ as the default mitigation candidate, while inserting the remaining sampled-but-unselected rows into the SHQ. For each default mitigation opportunity (TRR or RFM), PrISM mitigates and removes the highest-count PMQ entry. If the PMQ becomes full or any entry's count exceeds T_{PMQ}, PrISM asserts Alert to request an additional mitigation.

Third, intersecting rows may be serviced by default TRR/RFM opportunities before Alert becomes necessary, further reducing the Alert frequency and improving performance.

B. Operation of PrISM

Like MINT, PrISM operates on fixed-size mitigation windows. However, PrISM can use a longer default window W because it requests additional mitigations when repeated sampled activity creates intersections. Figure 3 illustrates the operation of PrISM. ❶ At the beginning of each mitigation window, PrISM randomly samples R activation slots out of the W slots and records each sampled-and-activated row in the SSQ. ❷ When a sampled activation occurs, PrISM checks whether the activated row matches any SHQ entry. ❸ If the row produces an intersection, PrISM enqueues it into the PMQ for additional mitigation. ❹ At the end of the window, PrISM randomly selects one non-intersecting sampled row from the SSQ and enqueues it into the PMQ as the default mitigation candidate. This ensures that each window contributes one default probabilistic mitigation, regardless of whether intersections occur. ❺ PrISM then updates the SHQ in FIFO order. Up to $R-1$ sampled-but-unselected rows from the current window are enqueued, while the oldest entries are evicted. If fewer than $R-1$ such rows exist because some sampled slots were never activated, PrISM inserts invalid placeholders to keep the lookback window L deterministic. ❻ At each default mitigation opportunity, such as a TRR during a refresh or an RFM, PrISM mitigates and removes the PMQ entry with the highest activation count. If the PMQ becomes full or any entry's counter exceeds T_{PMQ}, PrISM requests an additional mitigation via the Alert Back-Off (ABO) protocol, again servicing the entry with the highest counter.

Under benign workloads, intersections are infrequent because most rows are accessed sparsely and irregularly [78], [81], [101], [102]. In these common cases, the PMQ is mainly populated by default mitigation candidates, which are drained by upcoming default mitigations. PrISM therefore issues few Alerts and incurs negligible performance overhead.

C. JEDEC-Compatible Operation of PrISM

PrISM uses the existing JEDEC ABO protocol to request additional mitigations with *one* RFM per Alert. The key challenge is that Alert does not immediately drain a pending mitigation: after Alert, up to three activations (ABO_{ACT}) can occur before the corresponding RFM, and one ABO_{Delay} activation is required before Alert can be reasserted. Thus, under one RFM per Alert, ABO can drain only one pending entry every four activations, while the worst case can produce one intersection per activation. PrISM handles this by sizing the Sampled Slot Queue (SSQ) to temporarily buffer intersecting samples until PMQ entries are freed.

Figure 4 illustrates this worst case for $R=2$. Since PrISM samples R slots uniformly at random per window, all R slots can cluster near the boundary of two adjacent windows, and all may intersect with the SHQ, producing up to $2R$ intersections in $2R$ consecutive activations. The first intersection enters the PMQ and triggers Alert; the remaining intersections wait in the SSQ until ABO frees PMQ slots. Because ABO drains one pending entry every four activations, the SSQ must cover the peak number of waiting intersections during the burst. Of the $2R$–1 intersections after the first, $\lfloor (2R-1)/4 \rfloor$ can be drained before the burst completes. Thus, the required SSQ size is:

$$S_{\mathrm{SSQ}} \geq (2R-1) - \left\lfloor \frac{2R-1}{4} \right\rfloor \tag{1}$$

For our largest evaluated $R = 9$, this bound requires 13 SSQ entries. Beyond a single boundary burst, repeated bursts remain bounded as long as the long-run intersection rate does not exceed the ABO drain rate. Since at most R samples can intersect per W-activation window, $W \geq 4R$ is sufficient.

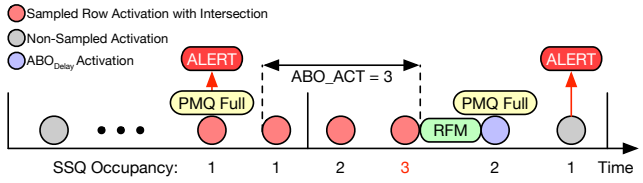

Fig. 4. Sampled slots can cluster near adjacent-window boundaries, and all intersect with the SHQ, producing intersections faster than ABO can drain. The SSQ buffers intersecting samples that cannot yet enter the PMQ.

D. Address Mapping for Ultra-Low $T_{RH\text{-}D}$

At ultra-low $T_{RH\text{-}D}$ (≤ 250), even benign workloads can create frequent intersections because conventional mappings preserve spatial locality. For example, Minimalist Open-Page (MOP) mapping [38] places nearby cache lines in the same DRAM row, improving row-buffer locality but also causing the same row to be sampled across multiple windows, reappear in the SHQ, and trigger unnecessary Alerts.

To reduce this overhead, PrISM uses randomized address mapping only for ultra-low $T_{RH\text{-}D}$ configurations. Randomized mapping spreads nearby cache lines across many DRAM rows, reducing persistent row-level locality. We use Rubix-style randomization [81], following prior work [70]. Table I shows the average per-channel Alert rate under both mappings. Randomization reduces benign Alerts across all thresholds, with the largest absolute reduction at $T_{RH\text{-}D} = 250$, where it cuts the Alert rate from 312.3 to 50.3 per 1K tREFI (6.2×). Thus, PrISM enables randomized mapping only at ultra-low $T_{RH\text{-}D}$ (≤ 250), while retaining conventional mapping at higher thresholds, where Alert rates are already low and preserving row-buffer locality is preferable.

TABLE I
AVERAGE PER-CHANNEL ALERT RATE OF PRISM

Target $T_{RH\text{-}D}$	MOP Alerts / 1K tREFI	Random Alerts / 1K tREFI	Reduction
1000	2.4	0.04	60×
500	32.2	3.5	9.2×
250	**312.3**	**50.3**	**6.2×**

IV. Security Analysis of PrISM

We analyze the security of PrISM against an attack pattern that maximizes the number of unmitigated activations to any aggressor row within a 32ms refresh window (tREFW). We quantify security using the *Mean-Time-to-Failure (MTTF)* metric. Unless otherwise stated, we target a per-bank MTTF of 10,000 years, comparable to the intrinsic DRAM soft-error rate [5], consistent with prior work [31], [70], [72], [98].

A. Worst-Case Attack: Circular-X-Rows Attack

The attacker must balance two competing goals. First, it wants to maximize the number of activations delivered to aggressor rows within tREFW. Second, it wants each aggressor to avoid PrISM's mitigations, including the default mitigation applied in every window and the additional intersection-based mitigations provided by SHQ. Repeated sampling increases the chance that a row is mitigated by the default mitigation or creates intersections that make the row eligible for additional mitigation. The worst-case pattern for PrISM is therefore the pattern that uses the full activation budget within tREFW while spreading activations across enough rows to minimize avoidable default selections and intersections.

We model this worst case using a *Circular-X-rows attack*. The attacker chooses X aggressor rows and repeatedly activates them in a round-robin manner, one row per activation slot. For a mitigation window of W activation slots, the row activated at slot s of window t is:

$$\text{row}(t, s) = (tW + s) \bmod X \tag{2}$$

The single parameter X controls the entire attack spectrum, exposing a fundamental trade-off between hammering rate per aggressor and evasion of SHQ intersections. Sweeping X over $[W, (L+1)W]$ captures all meaningful attacker strategies:

1) $X = W$**:** Every window contains W distinct rows hammered once each. This maximizes the activation rate per aggressor, but it also maximizes exposure to PrISM because the same rows repeatedly appear within the SHQ lookback window. This corresponds to the multi-row, single-copy pattern analyzed by MINT [72].
2) $W < X < (L+1)\,W$**:** The attacker spreads activations across more aggressor rows. This reduces the chance that a row reappears within the L-window SHQ history and therefore lowers the probability of intersections. However, it also reduces the number of activations delivered to each aggressor within tREFW. Thus, intermediate X values capture the fundamental tradeoff between hammering rate and mitigation exposure.
3) $X = (L+1)\,W$**:** The target row reappears exactly at the edge of the lookback window, completely evading SHQ intersections. Any larger X also evades intersections but reduces per-aggressor hammering rate by a factor of $\frac{X}{(L+1)W}$, weakening the attack. We therefore do not sweep beyond $X = (L+1)W$.

Three properties establish why the circular-X-rows pattern captures the worst case for PrISM.

1. Intra-window ordering is irrelevant: Because PrISM samples R slots uniformly at random from the W slots of a window, a row's sampling probability depends only on how many times it appears in the window, not on where those appearances fall [72]. Reordering activations within a window cannot change the probability of default selection, SHQ insertion, or intersection.

2. Using one copy per row per window is worst-case: If a row appears c times within a single window, its probability of being sampled at least once is:

$$P_{\text{sample}}(c) = 1 - \frac{\binom{W-c}{R}}{\binom{W}{R}} \tag{3}$$

$P_{\text{sample}}(c)$ increases with c, making the row more likely to be selected by the default mitigation, inserted into the SHQ, or causing additional mitigations through intersections. Additional copies of the same row also consume activation slots that could instead be used to attack other rows. Thus, to maximize the number of independent aggressors while minimizing avoidable sampling exposure, the attacker should activate each aggressor at most once per window.

3. Round-robin spacing is the strongest single-copy schedule: Under the single-copy constraint, the attacker's only remaining choice is how frequently to activate each row across windows. Clustering repeated activations of the same row increases the chance that the row remains within the L-window SHQ history, creating more intersections. Spreading activations avoids intersections, but reduces the per-aggressor hammering rate. Neither direction beats the optimal stationary spacing, so a periodic round-robin is optimal in the worst case.

B. Determining Minimum Supported $T_{RH\text{-}D}$ of PrISM

The security of PrISM is determined by *three* design parameters: the window size (W), the number of per-window sampled slots (R), and the lookback window (L). For each (W, R, L), we sweep $X \in [W, (L+1)W]$ and take the maximum required $\text{T}_{\text{RH-D}}$ as the security bound.

Per-window mitigation probability: For a circular-X-rows attack, let K denote the number of prior appearances of the same aggressor that fall within the SHQ's L lookback window (approximately $L \cdot W/X$). We model this SHQ residency probability with the following steady-state fixed point[3]:

$$P_{\text{SHQ}} = \frac{K\left(R - 1 + (P_{\text{SHQ}})^R\right)}{W + K \cdot R} \tag{4}$$

Furthermore, PrISM prioritizes rows *not* already present in the SHQ for default mitigation. Thus, the default mitigation contributes a new mitigation only when at least one of the R sampled rows is absent from the SHQ. The probability that all R sampled rows are already present in the SHQ is approximately $(P_{\text{SHQ}})^R$. Combining the effective default

[3]Appendix B provides the complete derivation for this occupancy model.

mitigation with the intersection-based mitigation, the per-window mitigation probability is:

$$P_m = \underbrace{\frac{1-(P_{\text{SHQ}})^R}{W}}_{\text{default mitigation}} + \underbrace{\frac{R}{W}\cdot P_{\text{SHQ}}}_{\text{intersection-based eligibility}} \quad (5)$$

We then apply the Saroiu-Wolman model [79] to convert P_m into the minimum $T_{RH\text{-}D}$ that meets our target MTTF (10,000 years), and sweep $X \in [W, (L+1)W]$ to find the worst-case X^* for each (W, R, L).

Impact of R and L at fixed W: Figure 5 shows the supported $T_{RH\text{-}D}$ as we vary the number of sampled activation slots (R) and the SHQ lookback window (L), at a window size of $W = 72$. Increasing either R or L improves security. A larger R increases the probability of sampling an aggressor in each window, increasing the chance that any hammered row is mitigated by the default mitigation or SHQ intersection. Larger L expands the history window, increasing the chance that an aggressor reappearing across windows produces an intersection. For example, PrISM can support $T_{RH\text{-}D}$ of 954 with $R = 3$ and $L = 25$, while raising R to 8 at the same L brings the supported $T_{RH\text{-}D}$ down to 494. We validate our analytical model using a 5-million-epoch Monte Carlo simulation that closely matches it.

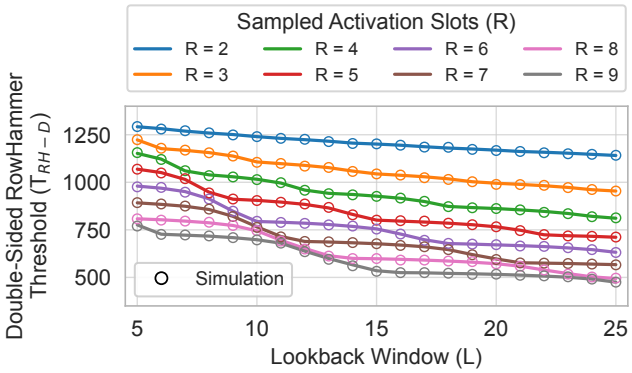

Fig. 5. Minimum supported $T_{RH\text{-}D}$ of PrISM under the worst-case circular-X-rows attack at $W = 72$, across sampled activation slots (R) and lookback windows (L). Larger R and L jointly increase the per-window mitigation probability, sharply lowering the supported $T_{RH\text{-}D}$.

Impact of W: Figure 6 shows the supported $T_{RH\text{-}D}$ as we vary the Sampled History Queue (SHQ) size and window size (W), with R swept subject to $W \geq 4R$ (Section III-C). Smaller W lowers the supported $T_{RH\text{-}D}$ at low SHQ capacities by raising both the default mitigation rate ($1/W$) and the probability that an aggressor is sampled into the SHQ. For example, at $\sim$40 SHQ entries, $W = 24$ supports $T_{RH\text{-}D}$ of 425, while $W = 72$ remains above 700. However, this benefit diminishes at larger SHQ sizes (≥ 150 entries), where the SHQ already provides enough history for persistent aggressors to intersect with high probability. Smaller W is therefore most useful for ultra-low $T_{RH\text{-}D}$ under a constrained SHQ budget.

Impact of T_{PMQ}: The analysis so far has assumed that a row selected for mitigation is mitigated immediately, yielding the base threshold $\widehat{T}_{RH\text{-}D}$. In PrISM, however, a selected row may remain in the Pending Mitigation Queue (PMQ) until it is serviced by a default mitigation (TRR or RFM) or through Alert Back-Off (ABO) protocol. While the row resides in the PMQ, it can accumulate further activations up to the *tardiness threshold* (T_{PMQ}) before Alert is asserted. Thus, the PMQ-aware supported threshold becomes $T_{RH\text{-}D} = \widehat{T}_{RH\text{-}D} + T_{PMQ}$. We use $T_{PMQ} = 4$ as the default.

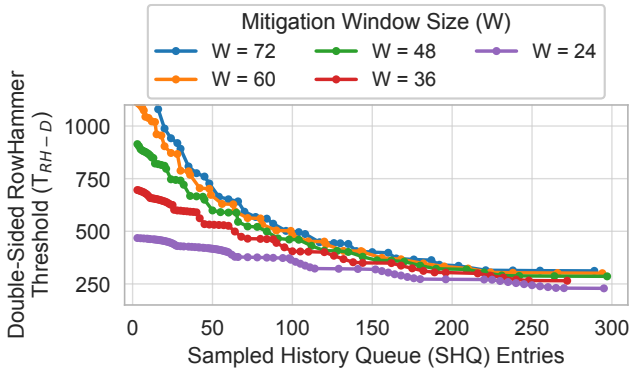

Fig. 6. Minimum Supported $T_{RH\text{-}D}$ of PrISM under the worst-case circular-X-rows attack with varying SHQ size and W. Smaller W lowers the supported $T_{RH\text{-}D}$ at low SHQ capacities, but its benefit diminishes once the SHQ provides enough history for repeated samples to intersect with high probability.

Impact of activations during Alert Back-Off (ABO_{ACT}): PrISM uses the existing JEDEC ABO protocol with one RFM per ABO. The protocol allows up to 3 activations (ABO_{ACT}) before the corresponding RFM is issued. Moreover, a subsequent Alert can only be raised once the controller receives the activations that match the number of RFMs during ABO. With a PMQ size of Q, an attacker can exploit these slack activations by preparing Q–1 entries near T_{PMQ}, then forcing a sequence of chained Alerts, analogous to the *Feinting* [58] and *Wave* [111] attacks against PRAC [71], [101]. This chain accumulates up to $ABO_{ACT}(Q)$ additional activations on a targeted row before it is mitigated. The supported threshold of PrISM therefore becomes $T_{RH\text{-}D} = \widehat{T}_{RH\text{-}D} + T_{PMQ} + ABO_{ACT}(Q)$. For our default PMQ size of 16, $ABO_{ACT}(Q)$ is 12, which combined with T_{PMQ} of 4 gives $T_{RH\text{-}D} = \widehat{T}_{RH\text{-}D} + 16$.

C. Determining PrISM Configurations

PrISM can support a target $T_{RH\text{-}D}$ through different combinations of (W, R, L). These parameters trade off performance and storage in different ways. Increasing R sharply lowers $T_{RH\text{-}D}$ by raising both per-window sampling and the probability of intersections, allowing PrISM to support lower $T_{RH\text{-}D}$ with fewer SHQ entries. However, larger R also increases the maximum number of possible intersections per window (R–1), which can raise Alert frequency and degrade performance. It can also make PrISM more vulnerable to potential *Denial-of-Service (DoS)* attacks, in which an adversary intentionally causes frequent intersections to trigger repeated Alerts (see Section VII for a detailed DoS analysis). Smaller W similarly reduces the required SHQ storage, particularly when the SHQ budget is tight, as shown in Figure 6. However,

a smaller W also leads to more frequent proactive RFMs, causing higher slowdown. We therefore choose the default configuration that balances performance and storage.

Table II shows the selected PrISM configurations. We use $W = 72$ whenever the target threshold can be met without an overly large R, preserving the lowest proactive RFM rate for $T_{RH\text{-}D} \geq 500$. For ultra-low $T_{RH\text{-}D}$ of 250, we reduce W to 48 to keep the SHQ size practical without relying on a large R. Overall, PrISM maintains practical hardware overhead. For $T_{RH\text{-}D}$ of 1000, it requires only a 36-entry SHQ, comparable to the TRR tracker already deployed in commercial DRAM [33]. Even at a $T_{RH\text{-}D}$ of 250, PrISM requires 632 entries, which remain 10–100× smaller than prior secure in-DRAM mitigations such as Mithril [46] and ProTRR [58] at the same threshold.

TABLE II
PRISM CONFIGURATIONS FOR TARGET ROWHAMMER THRESHOLDS

Target $T_{RH\text{-}D}$	Supported $T_{RH\text{-}D}$	Window Size (W)	Sampled Slots (R)	Lookback Window (L)	SHQ Entries $((R-1)\times L)$
1000	975	72	4	12	36
750	731	72	7	11	66
500	499	72	7	41	246
250	249	48	9	79	632

D. Sensitivity to Mean-Time-to-Failure (MTTF)

We next evaluate how the supported $T_{RH\text{-}D}$ of our chosen PrISM configurations changes under different MTTF targets. Table III reports the supported $T_{RH\text{-}D}$ for each configuration across a range of per-bank MTTF values, along with the corresponding system-level MTTF. We derive the system-level MTTF assuming an attacker targets up to 24 banks in parallel out of our 32-bank system, thereby maximizing the per-bank activation rate under the tFAW constraint. Targeting more banks would throttle the per-bank hammer rate and weaken the attack. PrISM provides strong protection across a wide MTTF range with practical SHQ sizes. For example, raising the per-bank MTTF target from 10K to 1M years (system MTTF from 417 to 41.7K years) shifts the supported $T_{RH\text{-}D}$ of the 66-entry configuration from 731 to 786, and the 36-entry configuration from 975 to 1069, a small relaxation in the supported threshold for a 100× improvement in MTTF.

TABLE III
SUPPORTED $T_{RH\text{-}D}$ OF PRISM WITH VARYING MEAN-TIME-TO-FAILURE

Target MTTF (Bank)	Target MTTF (System)	Selected PrISM Configuration			
		1K cfg. 36 SHQ	750 cfg. 66 SHQ	500 cfg. 246 SHQ	250 cfg. 632 SHQ
1K years	41.7 years	944	720	478	247
10K years	**417 years**	**975**	**731**	**499**	**249**
100K years	4.17K years	1017	747	507	262
1M years	41.7K years	1069	786	525	274

V. EVALUATION METHODOLOGY

Simulation Framework: We evaluate PrISM using Ramulator2 [49], [57], a cycle-accurate, trace-based DRAM simulator. Following prior work [68], [101], [102], [107], we employ Ramulator2's internal out-of-order (OoO) core model[4]. Our baseline system consists of an 8-core OoO processor, a 16MB shared LLC, and a single-channel, single-rank DDR5 configuration with 32GB of DRAM. The memory controller uses the Minimalist Open-Page address mapping [38] and a First-Ready First-Come First-Served scheduler [76], [115]. The DRAM is modeled as a 32Gb DDR5-8000B device with timing parameters from the JEDEC DDR5 specification [36]. Table IV shows the detailed system configuration.

TABLE IV
SYSTEM CONFIGURATION

Out-Of-Order Cores	8 Cores, 4GHz, 4-wide, 512-entry ROB
Last Level Cache (Shared)	16MB, 8-way, 32 MSHRs per core
Address Mapping	Minimalist Open-Page (MOP) [38]
Scheduling Policy	FR-FCFS [76], [115] with a cap of 1 [63]
Memory Type	32Gb DDR5-8000B
DRAM Organization	4 Bank × 8 Groups × 1 Rank × 1 Channel
Rows Per Bank, Size	128K, 8KB
tRCD, tCL, tRAS	16ns, 16ns, 32ns
tRP, tRTP, tWR, tRC	16ns, 7.5ns, 30ns, 48ns
tRFC, tREFI	410 ns, 3.9μs
$tRFM_{ab}$, $tRFM_{sb}$	350ns, 190ns

TABLE V
WORKLOAD CATEGORIZATION BASED ON RBMPKI

RBMPKI	Workloads
High (17) [10+)	429.mcf, 470.lbm, 434.zeusmp, 519.lbm, 549.fotonik3d, 459.GemsFDTD, 450.soplex, 462.libquantum, 433.milc, 437.leslie3d, 510.parest, 520.omnetpp, 483.xalancbmk, wc_8443, wc_map0, 482.sphinx3, tpch2
Medium (21) [1, 10)	471.omnetpp, grep_map0, 473.astar, 505.mcf, tpch17, 436.cactusADM, jp2_decode, 507.cactuBSSN, 557.xz, tpcc64, jp2_encode, ycsb_aserver, ycsb_eserver, ycsb_bserver, ycsb_cserver, ycsb_dserver, 500.perlbench, 523.xalancbmk, tpch6, ycsb_abgsave, 456.hmmer
Low (19) [0, 1)	401.bzip2, 502.gcc, 435.gromacs, 458.sjeng, 445.gobmk, 525.x264, 508.namd, 531.deepsjeng, 544.nab, 526.blender, 403.gcc, 464.h264ref, h264_encode, 447.dealII, 444.namd, 481.wrf, 541.leela, 538.imagick, 511.povray

Evaluated Design: We compare PrISM against two in-DRAM RowHammer mitigations: (1) Per Row Activation Counting (PRAC) [36] and (2) the state-of-the-art probabilistic scheme MINT [72]. For PRAC, we adopt QPRAC [101] as a secure baseline: each Alert triggers one RFM, and a 5-entry priority service queue tracks aggressor candidates for mitigation. We tune the Back-Off threshold for each target $T_{RH\text{-}D}$ following QPRAC's publicly available security model. For MINT, we follow its design with a delayed mitigation queue to support refresh and RFM postponement. The mitigation window size is set following the original work [72]: for $T_{RH\text{-}D}$ of 250, 500, and 1000, the window is 11, 24, and 48 activations, respectively.

For PrISM, we configure the mitigation window (W), the number of sampled slots (R), and lookback window (L) according to Table II for each target $T_{RH\text{-}D}$. We use a 16-entry Pending Mitigation Queue (PMQ) by default.

[4]We validated the internal core model by reproducing the PRAC interface-speed experiment in Figure 2 using the open-source ChampSim–Ramulator2 integration from prior work [103], [104]. Performance from the two setups agreed within 1.2% across 3200–8000MT/s DDR5 speeds and showed matching trends, so we report results using Ramulator2's internal core model.

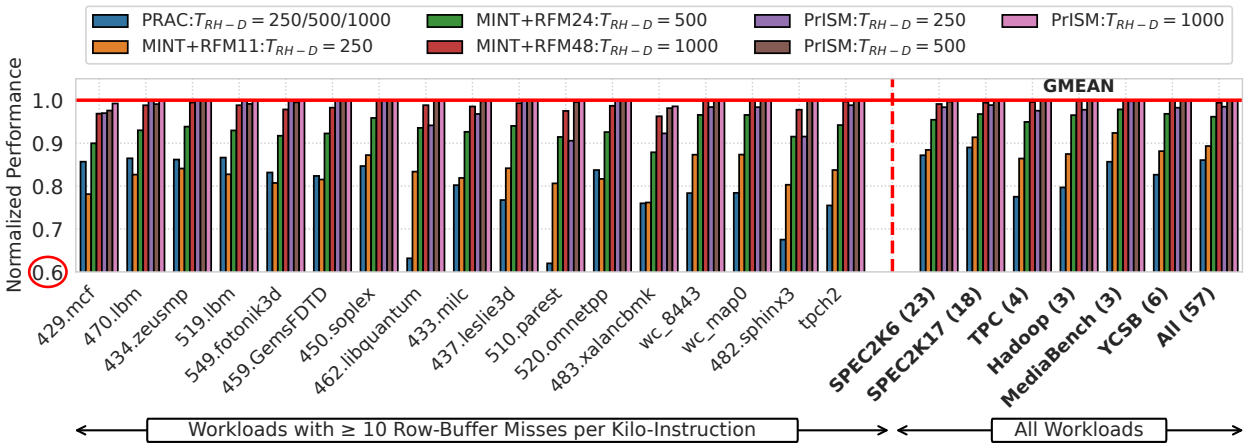

Fig. 7. Performance of PrISM at T_{RH-D} of 250, 500, and 1000, compared to MINT [72] and PRAC [101]. MINT incurs low slowdown at higher T_{RH-D}, but its slowdown increases as T_{RH-D} drops because lower thresholds require more frequent fixed-rate RFMs to address the non-selection problem. PRAC incurs roughly 14% average slowdown across thresholds because its overhead is dominated by inflated timing parameters. In contrast, PrISM incurs negligible slowdown ($\leq$0.2%) at $T_{RH-D} \geq 500$ and only 1.5% at ultra-low T_{RH-D} of 250, since intersections are infrequent on benign workloads.

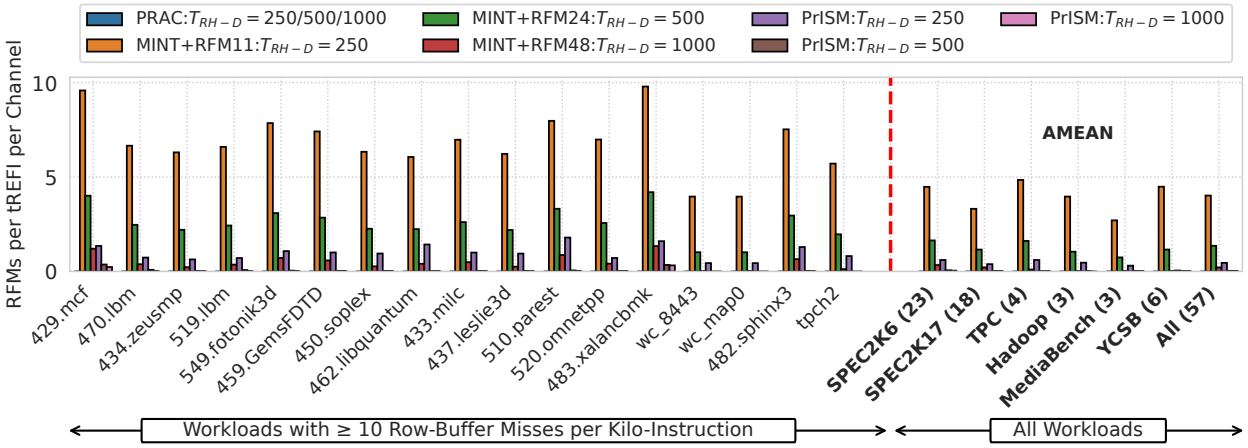

Fig. 8. RFM frequency per tREFI per channel for PrISM, MINT, and PRAC. MINT requires more frequent fixed-rate RFMs as T_{RH-D} drops, explaining its higher slowdown at low T_{RH-D}. PrISM issues fewer RFMs by keeping the default mitigation rate low and requesting additional RFMs through Alert Back-Off only on intersections. PRAC's precise per-row tracking yields negligible RFMs but still incurs overhead from inflated timing parameters.

Workloads: We evaluate 57 open-source workloads in Ramulator2 [77], drawn from SPEC2006/2017 [15], [89], TPC [95], Hadoop [18], MediaBench [20], and YCSB [14], classified by row-buffer misses per kilo-instruction (RBMPKI) as summarized in Table V. We run 8-core homogeneous mixes, with each core executing 250 million instructions for 2 billion instructions per run. By default, we issue one Target Row Refresh (TRR) every two refresh intervals and use Same-Bank RFM (RFM_{sb}) for proactive RFM. We also evaluate sensitivity to TRR frequency and PMQ size. Performance is reported as weighted speedup over a non-secure DDR5 baseline.

VI. Results and Analysis

A. Performance Overhead

Figure 7 compares the performance of PrISM, MINT, and PRAC across T_{RH-D} values of 250, 500, and 1000. PrISM achieves the lowest slowdown among the evaluated designs. For $T_{RH-D} \geq 500$, PrISM incurs negligible slowdown (under 1%), and only 1.5% even at the ultra-low T_{RH-D} of 250. The modest increase at low T_{RH-D} comes mainly from the shorter mitigation window, which increases the proactive RFM rate. As shown in Figure 8, PrISM issues only 0.03 RFMs per tREFI at T_{RH-D} of 500, and 0.45 at T_{RH-D} of 250.

MINT has low overhead at higher thresholds, but its overhead grows as T_{RH-D} drops because it raises its fixed mitigation rate to maintain security, reducing effective memory bandwidth. As Figure 8 shows, MINT issues only 0.2 RFMs per tREFI at T_{RH-D} of 1000, but 4 at T_{RH-D} of 250, leading to 3.8% and 10.7% average slowdown at T_{RH-D} of 500 and 250, respectively. The impact is most pronounced on high-memory-intensity workloads. For example, on *429.mcf*, MINT issues nearly 4 and 10 RFMs per tREFI at T_{RH-D} of 500 and 250, causing about 10% and 21.9% slowdown, respectively, while PrISM stays under 3.2% on the same workload across all thresholds. At T_{RH-D} of 250, MINT's slowdown even exceeds PRAC's on several workloads (*e.g., 470.lbm*, *434.zeusmp*, *520.omnetpp*). In contrast, PrISM's worst-case per-workload slowdown stays at 9.4% (*510.parest*) even at T_{RH-D} of 250.

PRAC issues virtually no RFMs thanks to its precise per-row tracking. However, its inflated timing parameters (tRP and tRC) from counter updates on every activation cause roughly 14% average slowdown across thresholds, exceeding 20% on several workloads such as *510.parest* and *wc_8443*. Overall, PrISM avoids the frequent fixed-rate RFMs required by MINT at low $T_{RH\text{-}D}$ and the timing overhead of PRAC, providing the lowest slowdown among the evaluated designs, especially for high-memory-intensity workloads at low $T_{RH\text{-}D}$.

B. Sensitivity to Target Row Refresh Ratio

Figure 9 compares the performance of PrISM, MINT, and PRAC as the Target Row Refresh (TRR) rate varies at $T_{RH\text{-}D}$ of 500. Lower TRR rates degrade both MINT and PrISM because they require more proactive RFMs rather than piggybacking mitigations onto TRR. With one TRR per tREFI, PrISM incurs negligible slowdown (<0.1%). Without TRR, PrISM issues an RFM roughly every 72 activations, increasing slowdown to 3.2%. Similarly, MINT's slowdown increases from 1.2% with one TRR per tREFI to 7.2% without TRR. In contrast, PRAC remains near 14% slowdown across TRR settings because its overhead is dominated by increased timing parameters rather than mitigation frequency. Overall, PrISM achieves the lowest slowdown across all TRR rates.

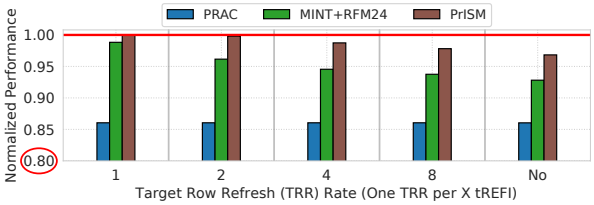

Fig. 9. Performance impact of TRR rate for PrISM, PRAC, and MINT at $T_{RH\text{-}D}$ of 500. Lower TRR rates increase reliance on proactive RFMs, degrading MINT and PrISM performance, while PRAC remains nearly constant because its slowdown stems primarily from increased timing parameters. Overall, PrISM achieves the lowest slowdown across all TRR settings.

C. Sensitivity to Pending Mitigation Queue Size

Table VI evaluates the impact of PrISM's Pending Mitigation Queue (PMQ) size at $T_{RH\text{-}D}$ of 500. A larger PMQ improves performance because rows selected through intersections can wait longer for default TRR or RFM opportunities, reducing the number of Alerts. However, it also increases storage and the worst-case activations an adversary can accumulate via delayed chained-Alert behavior (Section IV-B). Since performance saturates at a 16-entry PMQ (matching the 32-entry case at 0.2%), we use 16 entries by default.

TABLE VI
IMPACT OF PMQ SIZE ON $ABO_{ACT}(Q)$ AND PERFORMANCE

PMQ Size	$ABO_{ACT}(Q)$	Performance Overhead
4	7	0.6%
8	10	0.3%
16	**12**	**0.2%**
32	14	0.2%

D. Storage and Power Overhead

Storage: PrISM uses three per-bank structures: a Sampled History Queue (SHQ), a 13-entry Sampled Slot Queue (SSQ), and a 16-entry Pending Mitigation Queue (PMQ). Each SHQ and SSQ entry stores a 17-bit row address and a valid bit (18 bits total), and each PMQ entry additionally stores a 3-bit saturating activation counter. At $T_{RH\text{-}D}$ of 1000, PrISM requires a 36-entry SHQ, resulting in 152B of SRAM per bank; at $T_{RH\text{-}D}$ of 500, it requires 625B per bank.

This storage is substantially smaller than secure counter-based in-DRAM mitigations across all evaluated thresholds. At $T_{RH\text{-}D}$ of 1000, Mithril [46] requires 1082 27-bit entries even when issuing an RFM every 16 activations, the highest RFM rate defined in the DDR5 specification [36]; this is roughly **24**× more storage than PrISM. ProTRR [88] requires more than 16K entries per bank at the same threshold, or roughly **363**× more storage than PrISM. The gap narrows at lower thresholds but remains substantial: Mithril requires **20**× ($T_{RH\text{-}D}$ of 500) and **13**× ($T_{RH\text{-}D}$ of 250) more storage than PrISM, while ProTRR requires **170**× and **154**×, respectively. Thus, PrISM provides secure RowHammer protection with low slowdown while requiring one to two orders of magnitude less storage than secure counter-based in-DRAM mitigations.

Power: PrISM refreshes up to 4 victim rows per selected aggressor. Using the Micron power calculator [61], we estimate that these refreshes would account for 2.8% of DRAM power if issued every tREFI. However, even at $T_{RH\text{-}D}$ of 250, PrISM triggers Alert-induced mitigations only once every 20 tREFI on average, keeping refresh-related power overhead at 0.14%.

PrISM also requires random sampling, which can be implemented using an in-DRAM pseudo-random number generator (PRNG) or true random number generator (TRNG), as in prior in-DRAM probabilistic schemes [27], [31], [72]. Following MINT [72], we assume a 7-bit TRNG [39], [113] that consumes 90 μW of static and 200 μW of dynamic power (290 μW total), three orders of magnitude lower than DRAM chip power. We also estimate the dynamic power of PrISM's SRAM structures using CACTI-7.0 [4]: 1.5 mW at $T_{RH\text{-}D}$ of 500 and 2.4 mW at $T_{RH\text{-}D}$ of 250, corresponding to 0.6% and 1.0% of the 245 mW DRAM chip power estimated by the Micron power calculator [61]. Overall, PrISM incurs low power overhead. PrISM can also leverage recent in-DRAM TRNG designs [41], [65], [67], [94], which generate high-entropy bits from stochastic DRAM phenomena such as reduced-latency activation failures. PrISM's per-activation logic fits within tRC and incurs no DRAM timing overhead.

VII. ANALYZING DOS VULNERABILITY OF PRISM

PrISM uses the existing Alert Back-Off (ABO) protocol to request additional mitigations when repeated sampled activity creates SHQ intersections. Since ABO stalls an entire DRAM channel while RFMs are serviced, frequent Alerts can reduce bandwidth available to co-running applications. PrISM also performs default mitigations through periodic proactive RFMs, which consume additional bandwidth. An adversary can therefore exploit both sources of mitigation traffic to mount a

denial-of-service (DoS) attack, similar to prior performance attacks on RowHammer defenses [9], [64], [92], [102]. We now analyze PrISM's worst-case DoS exposure.

A. Attack Model and Assumptions

PrISM has low overhead on benign workloads because SHQ intersections are infrequent, so additional mitigations are requested only occasionally. Instead, a DoS adversary tries to maximize the number of intersections. We model this adversary using the Circular-X-Rows pattern from Section IV: the attacker repeatedly activates X rows in round-robin order so that sampled rows reappear in the SHQ and create frequent intersections. In the worst case, each mitigation window triggers one default RFM and up to $R-1$ additional RFMs from intersections, totaling at most R RFMs per W activations.

We assume proactive RFM_{ab} for default mitigation and disable Target Row Refresh (TRR), which is the worst case for PrISM because mitigations cannot be hidden within refresh time. Following prior work [71], [98], [101], we use activation throughput as the DoS metric. Let C_{RFM} denote the cost of one RFM_{ab} in activation-slot units:

$$C_{\text{RFM}} = \frac{t_{\text{RFMab}}}{t_{\text{RC}}} \tag{6}$$

With $t_{\text{RFMab}} = 350\text{ns}$ and $t_{\text{RC}} = 48\text{ns}$, one RFM_{ab} stalls roughly seven activations. Thus, if PrISM issues at most R RFMs per W activations, the worst-case throughput loss is:

$$\text{Throughput Loss} = \frac{C_{\text{RFM}} \cdot R}{W + C_{\text{RFM}} \cdot R} \approx \frac{7R}{W + 7R} \tag{7}$$

This bounds the worst-case throughput loss from default and intersection-induced mitigations under sustained attacks.

B. Worst-Case Slowdown under Performance Attacks

Table VII reports the worst-case slowdown of the selected PrISM configurations under the Circular-X DoS pattern. The default configurations incur worst-case slowdown factors of 1.39×, 1.68×, and 2.31× for $\text{T}_{\text{RH-D}}$ targets of 1000, 500, and 250, respectively. These slowdowns are comparable to existing memory performance attacks, such as conventional row-buffer conflict attacks [62], [63]. Thus, exploiting PrISM does not create a fundamentally more severe DoS vector than those already present in multi-core systems with shared memory.

TABLE VII
WORST-CASE DOS SLOWDOWN OF SELECTED PRISM CONFIGURATIONS

Target $\text{T}_{\text{RH-D}}$	W	R	Worst-Case Slowdown
1000	72	4	1.39×
500	72	7	1.68×
250	48	9	2.31×

This bound also exposes a performance-storage trade-off. Reducing R limits the maximum number of intersection-induced RFMs per window, improving DoS robustness. However, a smaller R also lowers the probability that persistent aggressors create SHQ intersections, so the system must increase the lookback window L to maintain the same security guarantee, increasing SHQ storage. Systems that prioritize stricter Quality-of-Service (QoS) guarantees can therefore use smaller-R, larger-L configurations, trading additional storage for lower worst-case slowdown.

VIII. RELATED WORK

A. Alert-Based RowHammer Mitigations

Several prior works use Alert for RowHammer mitigations. TWiCe [52] adds counters in the register clock driver and uses Alert to request mitigations from the memory controller, but incurs substantial area overhead even at a $\text{T}_{\text{RH-D}}$ of 5K. Self-Managing DRAM [23] and AutoRFM [70] leverage internal subarray structures to perform mitigations inside DRAM and redefine Alert to block activations to subarrays under mitigation. These approaches require significant changes to the DRAM core, limiting commercial adoption. In contrast, PrISM reuses the existing Alert Back-Off (ABO) protocol [36] and operates without DRAM array or interface modifications.

Concurrent work, MIRZA [90], also uses ABO to reduce the overhead of static-rate probabilistic mitigation. MIRZA uses per-subarray counters as a coarse-grained activity filter and enables probabilistic mitigation only when a subarray exceeds a predefined activation threshold. This reduces unnecessary mitigations, but its effectiveness depends on how rows are distributed across subarrays; MIRZA therefore proposes a strided row-to-subarray mapping to improve this distribution. In contrast, PrISM does not depend on a specific row-to-subarray mapping, since it correlates sampled row addresses directly across mitigation windows.

B. Comparison with Other PRAC Designs

Figure 10 compares PrISM with recent secure PRAC-based mitigations: MOAT [71], Chronus [10], and MoPAC [98]. We configure each design following its paper for each target $\text{T}_{\text{RH-D}}$. We assume one TRR opportunity per two tREFI intervals, and use two drains per tREFI for MoPAC.

The key difference is that all three prior designs remain fundamentally PRAC-based, relying on per-row activation counters. As a result, they inherit PRAC's area and power costs [27], [36], while requiring mechanisms to either perform or hide counter updates. MOAT deterministically updates counters on every activation and therefore inherits PRAC's timing bottleneck, causing a 14% average slowdown. Chronus avoids this bottleneck by updating counters concurrently with row accesses via a heterogeneous subarray architecture, resulting in negligible slowdown. However, Chronus requires DRAM core changes and can significantly tighten power constraints, potentially requiring a doubled tFAW budget [98].

MoPAC reduces counter update overhead by probabilistically updating counters. It samples a subset of activations, buffers the selected rows in a queue, and drains the pending counter updates through TRR opportunities or Alerts. However, as $\text{T}_{\text{RH-D}}$ drops, MoPAC must sample more frequently: roughly every 8 activations at $\text{T}_{\text{RH-D}}$ of 500 and every 4 activations at $\text{T}_{\text{RH-D}}$ of 250. This increases Alert frequency because the update queue fills more often, and drained updates

cause PRAC counters to reach the Alert threshold sooner. Consequently, MoPAC incurs 1.8% average slowdown at $T_{RH\text{-}D}$ of 500 and 6.5% at $T_{RH\text{-}D}$ of 250. In contrast, PrISM incurs only 1.5% average slowdown even at an ultra-low $T_{RH\text{-}D}$ of 250, while avoiding costly per-row activation counters entirely.

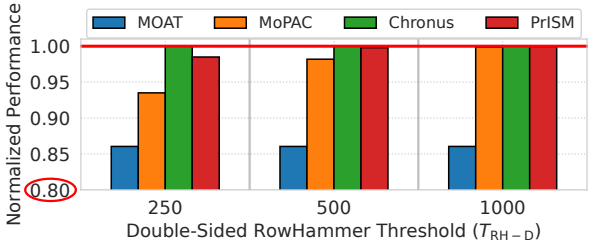

Fig. 10. Performance comparison of PrISM, MOAT [71], MoPAC [98], and Chronus [10]. MOAT incurs 14% slowdown from PRAC's per-activation counter updates, while MoPAC's slowdown grows at lower $T_{RH\text{-}D}$ due to more frequent sampled updates. PrISM incurs negligible slowdown ($\leq$0.2%) at $T_{RH\text{-}D} \geq 500$ and only 1.5% at $T_{RH\text{-}D}$ of 250, comparable to Chronus without requiring per-row counters or DRAM core changes.

C. Performance-Degradation Attacks and Mitigations

Recent work has shown that RowHammer mitigations can be exploited for performance-degradation attacks [64], [92]. DAPPER [102] employs secure hashing-based randomization, BreakHammer [9] throttles hardware threads that trigger excessive mitigation activity, and QPRAC [101] proposes to reduce interference from Alerts using finer-grained RFMs. Because PrISM primarily focuses on designing a secure, scalable probabilistic mitigation, its operation is orthogonal to these dedicated DoS defenses. These techniques are complementary to PrISM: access throttling can limit the triggers for adversarial mitigation, while finer-grained RFMs can reduce PrISM's mitigation cost caused by Alerts.

D. Host-Side RowHammer Mitigations

Host-side mitigations rely on the memory controller to select rows for mitigation and issue targeted refreshes, such as Directed RFM [36]. They either trigger mitigations probabilistically [40], [44], [48], [88], [91], [112] or use activation trackers to identify aggressor rows [8], [68], [69], [73], [82], [87]. Probabilistic schemes must increase mitigation frequency as RowHammer thresholds decrease, while tracker-based schemes require substantial storage to track many aggressors. In addition, host-side schemes may require additional support for transitive attacks, such as Half-Double [50], since the controller lacks proprietary internal DRAM mapping information. In contrast, PrISM performs mitigation selection inside DRAM using sampled history, enabling scalable protection at low thresholds without host-side row tracking.

E. Alternative Mitigation Mechanisms

Row-migration techniques [24], [78], [84], [99], such as SRS [105] and AQUA [84], mitigate RowHammer by relocating frequently activated rows. Access-throttling mechanisms [9], [107], such as BlockHammer [107], instead limit the activation rate of aggressive rows. However, these approaches can incur significant area or performance overhead as $T_{RH\text{-}D}$ drops below 1000. In contrast, PrISM preserves low slowdown at low thresholds by using in-DRAM sampled history to request additional mitigations only for repeated sampled activity.

F. ECC-Based RowHammer Defenses

ECC-based defenses [17], [47], [100] can tolerate some RowHammer-induced bit flips by correcting corrupted data after errors occur. However, modern DRAM devices can exhibit multiple bit flips within a single ECC codeword [33], [42], [108], exceeding the correction capability of conventional ECC. Prior work has also demonstrated successful RowHammer attacks on ECC-protected DRAMs [13], [37], including recent DDR5 devices [60]. In contrast, PrISM prevents RowHammer by mitigating aggressor rows before faults occur, providing protection even at ultra-low $T_{RH\text{-}D}$.

G. DRAM Architecture and Interface Redesigns

Other approaches propose fundamental redesigns of the DRAM architecture and memory interface to allow mitigations or refreshes to execute concurrently with DRAM activations [59], [110]. However, these schemes require modifications to the DRAM core and interface, hindering their commercial adoption. In contrast, PrISM achieves low-$T_{RH\text{-}D}$ protection by reusing the existing ABO protocol [36], avoiding DRAM array and interface changes.

H. Timing Channel Attacks on RowHammer Defenses

Recent work has shown that PRAC and RFM-based mitigations can introduce timing side channels because their RFM timing can depend on memory activity [7], [64], [104]. TPRAC [104] addresses this problem by making externally visible RFM timing independent of aggressor activity. Timing-channel protection is orthogonal to our goal, and PrISM can be composed with TPRAC-style techniques by decoupling mitigation selection from externally visible mitigation timing: intersections can still determine which rows enter the PMQ, while RFMs are released at fixed, activity-independent times.

IX. Conclusion

We presented PrISM, a scalable in-DRAM probabilistic mitigation that resolves the non-selection problem in prior probabilistic RowHammer defenses. PrISM correlates sampled rows across mitigation windows using a Sampled History Queue (SHQ), requesting additional mitigations only when repeated sampled activity creates intersections. This increases mitigation for persistent aggressors without uniformly increasing the default RFM rate. PrISM is compatible with the existing JEDEC Alert Back-Off protocol, requires no DRAM array changes, and avoids PRAC's per-row activation counters that must be updated on every activation. Our evaluations show that PrISM securely supports $T_{RH\text{-}D}$ of 500 with a negligible 0.2% average slowdown and 625B of SRAM per bank, and scales to an ultra-low $T_{RH\text{-}D}$ of 250 with 1.5% average slowdown. Overall, PrISM improves low-threshold scalability over fixed-rate probabilistic defenses while providing lower performance overhead than PRAC and one-to-two orders of magnitude less storage than secure counter-based in-DRAM mitigations.

Acknowledgment

We thank the anonymous ISCA 2026 reviewers for their valuable comments. We are especially grateful to Moinuddin Qureshi and Reviewer-F for their detailed feedback, which significantly strengthened the final version of this paper. We also thank the Advanced Research Computing (ARC) team at the University of British Columbia (UBC) for their computing support [1]. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under funding number RGPIN-2019-05059. The views expressed are those of the authors and do not necessarily reflect those of NSERC, NVIDIA, UBC, or the Government of Canada.

Appendix A
PRAC Performance Comparison

Prior work [43] evaluated PRAC on a real DDR5-4800 system and reported a 1.3% slowdown on SPEC2017, whereas we observe an average 3.7% slowdown at the same interface speed. This difference mainly stems from workload intensity and system configuration. First, our evaluated SPEC2017 workloads have higher RBMPKI, making them more sensitive to PRAC's increased timing parameters. Second, the prior study used 8 P-cores across two memory channels, corresponding to 4 cores per channel, whereas our setup uses 8 cores in a single channel. This 8-core-per-channel configuration is closer to modern server-class CPUs, which commonly provide roughly 8–16 physical cores per memory channel [2], [29]. Finally, the prior system provides 8.5MB of effective cache capacity per core, substantially reducing memory traffic and masking PRAC overhead. In contrast, we provision a server-representative 2MB of LLC per core [2], [29]. Recent studies [11], [90] make similar observations.

Appendix B
Analytical Model of SHQ Occupancy

This appendix derives the steady-state SHQ-presence probability used in Section IV. The model captures whether a repeatedly activated row is present in the Sampled History Queue (SHQ), which determines whether a future sampled activation creates an intersection. We use a two-state discrete-time Markov chain: S_0 denotes that the row is absent from the SHQ, and S_1 denotes that the row is present.

A transition from S_0 to S_1 occurs when the row is sampled in a mitigation window but is not selected as the default mitigation candidate, so it is inserted into the SHQ. For a row that appears once in a W-activation window, PrISM samples the row with probability R/W. Given that it is sampled, one of the R sampled rows is selected for default mitigation, so the row is inserted into the SHQ with probability $(R-1)/R$. Thus, the effective per-window insertion probability is:

$$P_{\text{in}} = \frac{R}{W} \cdot \frac{R-1}{R} = \frac{R-1}{W} \tag{8}$$

A transition from S_1 to S_0 occurs when the row's SHQ entry expires. Since each entry persists for L lookback windows, we approximate the per-window exit probability as $1/L$.

At steady state, the insertion and eviction rates are equal:

$$P(S_0) \cdot P_{\text{in}} = P(S_1) \cdot \frac{1}{L}.$$

Using $P(S_0) = 1 - P(S_1)$ and denoting $P(S_1) = P_{\text{SHQ}}$, we obtain:

$$P_{\text{SHQ}} = \frac{L \cdot P_{\text{in}}}{1 + L \cdot P_{\text{in}}}. \tag{9}$$

This model gives the steady-state SHQ residency probability for a repeated aggressor. We use this value as P_{SHQ} in Section IV to compute intersection-based mitigation probability.

Appendix C
Extending PrISM to RowPress and ColumnDisturb

PrISM can be extended to support other read-disturbance mechanisms such as RowPress [56] and ColumnDisturb [114].

RowPress [56]: PrISM can mitigate RowPress by adopting the ImPress [80] technique, as also used by MINT [72]. Each row-open interval is converted into an equivalent activation count, $EACT = (t_{ON} + t_{PRE})/t_{RC}$. PrISM then advances its per-window activation counter by $EACT$ instead of one. If this increment crosses a sampled slot, the open row is explicitly sampled. This allows PrISM to capture time-based aggressors using the same SHQ-intersection logic.

ColumnDisturb [114]: ColumnDisturb can disturb rows across the aggressor subarray and neighboring subarrays, so mitigation requires refreshing many potential victim rows rather than only adjacent rows [114]. PrISM can serve as an efficient filter for identifying repeatedly activated aggressors and triggering mitigation mechanisms such as Proactively Refreshing Victim Rows (PRVR) [114].

References

[1] "UBC Advanced Research Computing, "UBC ARC Sockeye." UBC Advanced Research Computing, 2019, doi: 10.14288/SOCKEYE."

[2] AMD. (2024) AMD EPYC 9965 Processor. Accessed: 2025-11-10. [Online]. Available: https://www.amd.com/en/products/processors/server/epyc/9005-series/amd-epyc-9965.html

[3] S. Baek, M. Wi, S. Park, H. Nam, M. J. Kim, N. S. Kim, and J. H. Ahn, "Marionette: A rowhammer attack via row coupling," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 637–652. [Online]. Available: https://doi.org/10.1145/3669940.3707242

[4] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3085572

[5] M. V. Beigi, Y. Cao, S. Gurumurthi, C. Recchia, A. Walton, and V. Sridharan, "A systematic study of ddr4 dram faults in the field," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 991–1002.

[6] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A complete in-dram rowhammer mitigation," in *Workshop on DRAM Security (DRAMSec)*, 2021.

[7] F. N. Bostancı, O. Canpolat, A. Olgun, I. E. Yüksel, K. Kanellopoulos, M. Sadrosadati, A. G. Yağlıkçı, and O. Mutlu, "Understanding and mitigating covert channel and side channel vulnerabilities introduced by rowhammer defenses," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1412–1432. [Online]. Available: https://doi.org/10.1145/3725843.3756029

[8] F. N. Bostanci, I. E. Yüksel, A. Olgun, K. Kanellopoulos, Y. C. Tuğrul, A. G. Yağliçi, M. Sadrosadati, and O. Mutlu, "Comet: Count-min-sketch-based row tracking to mitigate rowhammer at low cost," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 593–612.

[9] O. Canpolat, A. G. Yağlıkçı, A. Olgun, I. E. Yuksel, Y. C. Tuğrul, K. Kanellopoulos, O. Ergin, and O. Mutlu, "Breakhammer: Enhancing rowhammer mitigations by carefully throttling suspect threads," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 915–934.

[10] 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 *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 887–905.

[11] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, N. Bostancı, İsmail Emir Yüksel, H. Luo, O. Ergin, and O. Mutlu, "Chronus: Understanding and securing the cutting-edge industry solutions to dram read disturbance," 2025. [Online]. Available: https://arxiv.org/abs/2502.12650

[12] Z. Coalson, J. Woo, S. Chen, Y. Sun, L. Yang, P. Nair, B. Fang, and S. Hong, "Prisonbreak: Jailbreaking large language models with fewer than twenty-five targeted bit-flips," 2024. [Online]. Available: https://arxiv.org/abs/2412.07192

[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] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 143–154. [Online]. Available: https://doi.org/10.1145/1807128.1807152

[15] S. P. E. Corporation, "Spec cpu2006 benchmark suite," 2006. [Online]. Available: http://www.spec.org/cpu2006/

[16] 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

[17] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 373–386.

[18] A. Foundation, "Apache hadoop." [Online]. Available: http://hadoop.apache.org/

[19] P. Frigo, E. Vannacc, H. Hassan, V. v. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 747–762.

[20] 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.*, vol. 33, no. 4, p. 301–318, Jun. 2009. [Online]. Available: https://doi.org/10.1016/j.micpro.2009.02.010

[21] 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 *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 245–261.

[22] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, ser. DIMVA 2016. Berlin, Heidelberg: Springer-Verlag, 2016, p. 300–321. [Online]. Available: https://doi.org/10.1007/978-3-319-40667-1_15

[23] H. Hassan, A. Olgun, A. G. Yağlıkçı, H. Luo, O. Mutlu, and E. Zurich, "Self-managing dram: a low-cost framework for enabling autonomous and efficient dram maintenance operations," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 949–965.

[24] 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*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 129–142. [Online]. Available: https://doi.org/10.1145/3307650.3322231

[25] 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," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1198–1213. [Online]. Available: https://doi.org/10.1145/3466752.3480110

[26] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 497–514. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/hong

[27] S. Hong, D. Kim, J. Lee, R. Oh, C. Yoo, S. Hwang, and J. Lee, "Dsac: Low-cost rowhammer mitigation using in-dram stochastic and approximate counting algorithm," *arXiv:2302.03591*, 2023.

[28] 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 *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

[29] Intel Corporation. (2026) Intel® Xeon® 6756P-B Processor Specifications. Accessed: 2026-05-01. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/245156/intel-xeon-6756pb-processor-256m-cache-2-20-ghz/specifications.html

[30] A. Jaleel, S. W. Keckler, and G. Saileshwar, "Probabilistic tracker management policies for low-cost and scalable rowhammer mitigation," *arXiv:2404.16256*, 2024.

[31] A. Jaleel, G. Saileshwar, S. W. Keckler, and M. Qureshi, "Pride: Achieving secure rowhammer mitigation with low-cost in-dram trackers," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1157–1172.

[32] 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

[33] 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.

[34] 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

[35] JEDEC, "Ddr4 sdram standard (jesd79-4b)," 2017.

[36] JEDEC. (2024) JESD79-5C. https://www.jedec.org/document_search?search_api_views_fulltext=jesd79-5c.

[37] N. Kamadan, W. Wang, S. van Schaik, C. Garman, D. Genkin, and Y. Yarom, "{ECC. fail}: Mounting rowhammer attacks on {DDR4} servers with {ECC} memory," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 5679–5698.

[38] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: a dram page-mode scheduling policy for the many-core era," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, p. 24–35. [Online]. Available: https://doi.org/10.1145/2155620.2155624

[39] O. Katz, D. A. Ramon, and I. A. Wagner, "A robust random number generator based on a differential current-mode chaos," *IEEE Transac-*

tions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 12, pp. 1677–1686, 2008.

[40] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 9–12, 2015.

[41] 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 *2019 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 582–595.

[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 *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA '20. IEEE Press, 2020, p. 638–651. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00059

[43] J. Kim, S. Baek, M. Wi, H. Nam, M. J. Kim, S. Lee, K. Sohn, and J. H. Ahn, "Per-row activation counting on real hardware: Demystifying performance overheads," *IEEE Computer Architecture Letters*, vol. 24, no. 2, pp. 217–220, 2025.

[44] K. Kim, J. Woo, J. Kim, and K.-S. Chung, "Hammerfilter: Robust protection and low hardware overhead method for rowhammer," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 212–219.

[45] M. J. Kim, S. Baek, J. Kim, H. Nam, N. S. Kim, and J. H. Ahn, "Sok: Systematizing a decade of architectural rowhammer defenses through the lens of streaming algorithms," in *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

[46] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative row hammer protection on commodity dram leveraging managed refresh," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1156–1169.

[47] M. J. Kim, M. Wi, J. Park, S. Ko, J. Choi, H. Nam, N. S. Kim, J. H. Ahn, and E. Lee, "How to kill the second bird with one ecc: The pursuit of row hammer resilient dram," in *MICRO*, 2023.

[48] 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," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, p. 361–372, Jun. 2014. [Online]. Available: https://doi.org/10.1145/2678373.2665726

[49] 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.

[50] 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

[51] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 695–711.

[52] 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

[53] C. S. Lin, J. Qu, and G. Saileshwar, "Gpuhammer: Rowhammer attacks on gpu memories are practical," in *USENIX Security*, 2025.

[54] C. S. Lin, Y. Yan, G. Ding, J. Qu, J. Zhu, D. Lie, and G. Saileshwar, "GPUBreach: Privilege escalation attacks on GPUs using rowhammer," in *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

[55] 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

[56] 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*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589063

[57] 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," *IEEE Comput. Archit. Lett.*, vol. 23, no. 1, p. 112–116, Jan. 2024. [Online]. Available: https://doi.org/10.1109/LCA.2023.3333759

[58] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "Protrr: Principled yet optimal in-dram target row refresh," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 735–753.

[59] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "Rega: Scalable rowhammer mitigation with refresh-generating activations," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1684–1701.

[60] D. Meyer, P. Jattke, M. Marazzi, S. Qazi, D. Moghimi, and K. Razavi, "Phoenix: Rowhammer attacks on ddr5 with self-correcting synchronization," in *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

[61] Micron Technology Inc., "System Power Calculators," https://www.micron.com/support/tools-and-utilities/power-calc.

[62] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in Multi-Core systems," in *16th USENIX Security Symposium (USENIX Security 07)*. Boston, MA: USENIX Association, Aug. 2007. [Online]. Available: https://www.usenix.org/conference/16th-usenix-security-symposium/memory-performance-attacks-denial-memory-service-multi

[63] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 146–160.

[64] R. Nazaraliyev, Y. Zhang, S. B. Dutta, A. Marquez, K. Barker, and N. Abu-Ghazaleh, "Not so refreshing: Attacking {GPUs} using {RFM} rowhammer mitigation," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 5641–5660.

[65] A. Olgun, F. N. Bostanci, O. Canpolat, G. F. Oliveira, M. Sadrosadati, A. G. Yağlikçi, O. Mutlu *et al.*, "In-dram true random number generation using simultaneous multiple-row activation: An experimental study of real dram chips," in *2025 IEEE 43rd International Conference on Computer Design (ICCD)*. IEEE, 2025, pp. 754–763.

[66] A. Olgun, M. Osseiran, A. G. Yağlıkçı, Y. C. Tuğrul, 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 *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024, pp. 75–89.

[67] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, F. 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 *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 944–957.

[68] 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," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1579–1596. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/olgun

[69] 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.

[70] M. Qureshi, "Autorfm: Scaling low-cost in-dram trackers to ultra-low rowhammer thresholds," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 991–1004.

[71] M. Qureshi and S. Qazi, "Moat: Securely mitigating rowhammer with per-row activation counters," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 698–714. [Online]. Available: https://doi.org/10.1145/3669940.3707278

[72] M. Qureshi, S. Qazi, and A. Jaleel, "Mint: Securely mitigating rowhammer with a minimalist in-dram tracker," in *2024 57th IEEE/ACM*

International Symposium on Microarchitecture (MICRO), 2024, pp. 899–914.

[73] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 699–710. [Online]. Available: https://doi.org/10.1145/3470496.3527421

[74] M. K. Qureshi, "Salt: Track-and-mitigate subarrays, not rows, for blast-radius-free rowhammer defense," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2026, pp. 1–16.

[75] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[76] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 2000, pp. 128–138.

[77] SAFARI Research Group, "ABACuS — GitHub Repository," 2023. [Online]. Available: https://github.com/CMU-SAFARI/ABACuS

[78] 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

[79] S. Saroiu and A. Wolman, "How to configure row-sampling-based rowhammer defenses," in *Workshop on DRAM Security*, 2022.

[80] A. Saxena, A. Jaleel, and M. Qureshi, "Impress: Securing dram against data-disturbance errors via implicit row-press mitigation," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 935–948.

[81] A. Saxena, S. Mathur, and M. Qureshi, "Rubix: Reducing the overhead of secure rowhammer mitigations via randomized line-to-row mapping," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1014–1028. [Online]. Available: https://doi.org/10.1145/3620665.3640404

[82] A. Saxena and M. Qureshi, "Start: Scalable tracking for any rowhammer threshold," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 578–592.

[83] A. Saxena, G. Saileshwar, J. Juffinger, A. Kogler, D. Gruss, and M. Qureshi, "Pt-guard: Integrity-protected page tables to defend against breakthrough rowhammer attacks," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023, pp. 95–108.

[84] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 108–123.

[85] A. Saxena, W. Wang, and A. Daglis, "Citadel: Rethinking memory allocation to safeguard against inter-domain rowhammer exploits," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1117–1131. [Online]. Available: https://doi.org/10.1145/3725843.3756098

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

[87] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating wordline crosstalk using adaptive trees of counters," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 612–623. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00057

[88] M. Son, H. Park, J. Ahn, and S. Yoo, "Making dram stronger against row hammering," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[89] "SPEC CPU2017 Benchmark Suite," Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2017/

[90] H. Taneja, A. Hajiabadi, M. Marazzi, K. Razavi, and M. Qureshi, "Mirza: Efficiently mitigating rowhammer with randomization and alert," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2026, pp. 1–13.

[91] H. Taneja and M. Qureshi, "Dream: Enabling low-overhead rowhammer mitigation via directed refresh management," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 776–792. [Online]. Available: https://doi.org/10.1145/3695053.3731117

[92] H. Taneja and M. Qureshi, "Roguerfm: Attacking refresh management for covert-channel and denial-of-service," 2025. [Online]. Available: https://arxiv.org/abs/2501.06646

[93] TechPowerUp. (2024, October) DDR6 Memory Arrives in 2027 with 8,800–17,600 MT/s Speeds. Accessed: 2025-11-10. [Online]. Available: https://www.techpowerup.com/339178/ddr6-memory-arrives-in-2027-with-8-800-17-600-mt-s-speeds

[94] F. Tehranipoor, W. Yan, and J. A. Chandy, "Robust hardware true random number generators using dram remanence effects," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2016, pp. 79–84.

[95] Transaction Processing Performance Council, "TPC Benchmarks." [Online]. Available: http://tpc.org/

[96] Y. C. Tuğrul, A. G. Yağlıkçı, İ. E. Yüksel, A. Olgun, O. Canpolat, N. Bostancı, 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 *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 867–886.

[97] 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, p. 1675–1689. [Online]. Available: https://doi.org/10.1145/2976749.2978406

[98] S. Vittal, S. Qazi, P. Das, and M. Qureshi, "Mopac: Efficiently mitigating rowhammer with probabilistic activation counting," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 723–738. [Online]. Available: https://doi.org/10.1145/3695053.3730997

[99] M. Wi, J. Park, S. Ko, M. J. Kim, N. Sung Kim, E. Lee, and J. H. Ahn, "Shadow: Preventing row hammer in dram with intra-subarray row shuffling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 333–346.

[100] M. Wi, Y. Yoo, Y. Kim, J. Shin, J. Kim, Y. Ryu, S. Gorgin, J. H. Ahn, and J. Kim, "Rowarmor: Efficient and comprehensive protection against dram disturbance attacks," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 1640–1659. [Online]. Available: https://doi.org/10.1145/3779212.3790213

[101] J. Woo, S. C. Lin, P. J. Nair, A. Jaleel, and G. Saileshwar, "Qprac: Towards secure and practical prac-based rowhammer mitigation using priority queues," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1021–1037.

[102] J. Woo and P. J. Nair, "Dapper: A performance-attack-resilient tracker for rowhammer defense," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1005–1020.

[103] J. Woo, J. Qu, G. Saileshwar, and P. J. Nair, "Artifact for when mitigations backfire: Timing channel attacks and defense for prac-based rowhammer mitigations," 2025, [Online]. Available: https://github.com/STAR-Laboratory/PRAC_TC_ISCA25.

[104] J. Woo, J. Qu, G. Saileshwar, and P. J. Nair, "When mitigations backfire: Timing channel attacks and defense for prac-based rowhammer mitigations," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 739–756. [Online]. Available: https://doi.org/10.1145/3695053.3731007

[105] 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.

[106] R. Wu, M. Zhang, Y. Zhou, C. Xie, and F. Wu, "Apt: Securing against dram read disturbance via adaptive probabilistic in-dram trackers," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 137–156. [Online]. Available: https://doi.org/10.1145/3779212.3790126

[107] 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.

[108] A. G. Yağlıkçı, Y. C. Tuğrul, G. F. Oliveira, İ. E. Yüksel, 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 *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 560–577.

[109] F. Yao, A. S. Rakin, and D. Fan, "DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1463–1480. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yao

[110] A. G. Yağlikçi, 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 *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 815–834.

[111] A. G. Yağlıkçı, J. S. Kim, F. Devaux, and O. Mutlu, "Security analysis of the silver bullet technique for rowhammer prevention," 2021. [Online]. Available: https://arxiv.org/abs/2106.07084

[112] J. M. You and J.-S. Yang, "Mrloc: Mitigating row-hammering based on memory locality," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[113] F. Yu, L. Li, Q. Tang, S. Cai, Y. Song, and Q. Xu, "A survey on true random number generators based on chaos," *Discrete Dynamics in Nature and Society*, vol. 2019, no. 1, p. 2545123, 2019.

[114] I. E. Yuksel, A. Olgun, N. Bostanci, H. Luo, A. G. Yaglikci, and O. Mutlu, "Columndisturb: Understanding column-based read disturbance in real dram chips and implications for future systems," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 975–994. [Online]. Available: https://doi.org/10.1145/3725843.3756022

[115] W. K. Zuravleff and T. Robinson, "Controller for a synchronous dram that maximizes throughput by allowing memory requests and commands to be issued out of order," May 13 1997, uS Patent 5,630,096.