2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

# QPRAC: Towards Secure and Practical PRAC-based Rowhammer Mitigation using Priority Queues

Jeonghyun Woo[†,§], Shaopeng (Chris) Lin*, Prashant J. Nair[†], Aamer Jaleel[‡], Gururaj Saileshwar*
[†]University of British Columbia, [‡]NVIDIA, *University of Toronto
jhwoo36@ece.ubc.ca, shaopenglin@cs.toronto.edu, prashantnair@ece.ubc.ca, ajaleel@nvidia.com, gururaj@cs.toronto.edu

***Abstract*—JEDEC has introduced the Per Row Activation Counting (PRAC) framework for DDR5 and future DRAMs to enable precise counting of DRAM row activations. PRAC enables a holistic mitigation of Rowhammer attacks even at ultra-low Rowhammer thresholds. PRAC uses an Alert Back-Off (ABO) protocol to request the memory controller to issue Rowhammer mitigation requests. However, recent PRAC implementations are either insecure or impractical. For example, Panopticon, the inspiration for PRAC, is rendered insecure if implemented per JEDEC's PRAC specification. On the other hand, the recent UPRAC proposal is impractical since it needs oracular knowledge of the 'top-N' activated DRAM rows that require mitigation.**

**This paper provides the first secure, scalable, and practical RowHammer solution using the PRAC framework. The crux of our proposal is the design of a priority-based service queue (PSQ) for mitigations that prioritizes pending mitigations based on activation counts to avoid the security risks of prior solutions. This provides principled security using the reactive ABO protocol. Furthermore, we co-design our PSQ, with opportunistic mitigation on Refresh Management (RFM) operations and proactive mitigation during refresh (REF), to limit the performance impact of ABO-based mitigations. QPRAC provides secure and practical RowHammer mitigation that scales to Rowhammer thresholds as low as 71 while incurring a 0.8% slowdown for benign workloads, which further reduces to 0% with proactive mitigations.**

## I. Introduction

Relentless scaling of Dynamic Random Access Memory (DRAM) technology has exposed critical security vulnerabilities like Rowhammer (RH). RH exploits inter-cell interference to rapidly activate DRAM rows, causing bit-flips in neighboring victim rows [1], [6], [11], [13], [14], [55], [61]. The number of activations needed to induce bit-flips, known as the Rowhammer threshold ($T_{RH}$), has dropped from 70K [29] to 4.8K [24] and is expected to decrease further with each generation. To counteract RH, the DRAM industry has proposed a series of in-DRAM mitigations, with the latest being Per Row Activation Counting (PRAC) [40]. However, the PRAC specification provides minimal implementation details, placing significant responsibility on DRAM manufacturers. This paper introduces a solution to implement PRAC securely and practically in DRAM for ultra-low $T_{RH}$ values (sub-100).

Prior in-DRAM RH mitigations implemented by DRAM vendors commercially have repeatedly fallen short in either security or scalability. For example, DDR4 devices use Targeted Row Refresh (TRR), which relies on a tracker to identify aggressor rows and refresh neighboring victim rows [16]. However, these trackers can only monitor a limited number of rows and are vulnerable to attack patterns like TRRespass, which target a larger number of rows [11]. DDR5 introduced the Refresh Management (RFM) command to mitigate victim rows proactively. This limits the number of activations per bank before an RFM-based mitigation needs to be issued. However, such solutions do not scale to $T_{RH}$ below 100. Even state-of-the-art defenses like PrIDE [19] and MINT [47] require frequent RFMs (e.g., 1 RFM every 10 activations), resulting in nearly 30% activation bandwidth loss at $T_{RH}$ of 250. Consequently, JEDEC, the DRAM standards committee, proposed PRAC for DDR5 DRAM chips (and beyond) [40].

PRAC maintains activation counters for each row in DRAM and allows the DRAM to use the Alert Back-Off (ABO) protocol to request an RFM from the host only when mitigation is needed. The ABO protocol uses the *Alert_n* pin in the DRAM module to notify the memory controller when any row activation exceeds the Back-Off threshold ($N_{BO}$), which is set lower than the $T_{RH}$. This prompts the memory controller to issue RFM commands on demand and perform RH mitigation before a row reaches $T_{RH}$. Although recent work explored this approach for ultra-low $T_{RH}$ (sub-100), they face security concerns or impractical overheads.

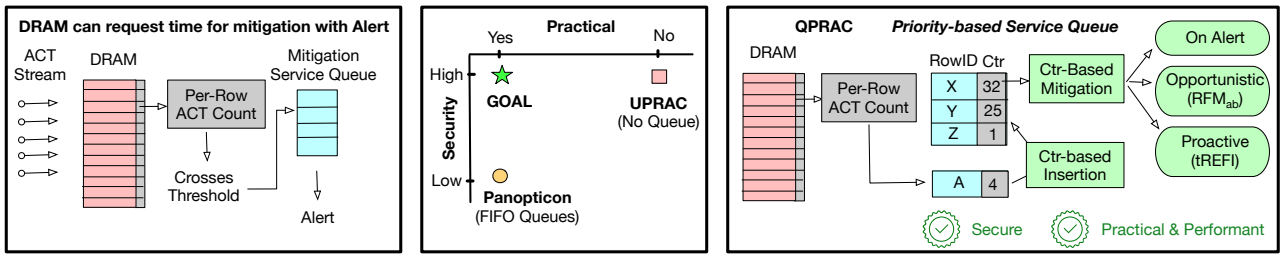
**1. Lack of Security**: Panopticon [2], the inspiration behind PRAC, also uses in-DRAM per-row activation counters and a FIFO-based service queue to track rows exceeding an activation threshold. When the queue is full, the DRAM module uses the ABO protocol to stop activations and request an RFM for RH mitigation, thus freeing up space in the queue.

However, Panopticon is insecure under the PRAC specification. PRAC employs a non-blocking ABO protocol, allowing the memory controller to continue issuing additional activations for up to 180ns. This window permits a row to surpass the mitigation threshold while bypassing the service queue if the queue is already full. Moreover, Panopticon only selects a row for mitigation when the threshold bit (t) in its counter toggles. Thus, the next insertion for a bypassed row occurs only after $2^t$ activations. We show that this leads to unmitigated rows being activated up to 50× higher than $T_{RH}$, compromising security.

**2. Impractical Overheads**: UPRAC [4] proposes a PRAC implementation *without* a service queue. This design triggers an Alert when any DRAM row exceeds $N_{BO}$. Thereafter, it mitigates the top-N activated rows with N subsequent RFMs.

While this avoids the security issues of service queues, it incurs impractical performance overheads. During each Alert, a bank must search through activation counters for *all* rows to identify the top-N rows, which is impractical. For example,

[§]A large part of this work was performed while Jeonghyun Woo was interning with NVIDIA Research.




(a) DDR5 specification introduces Per Row Activation Counting (PRAC)

(b) Existing PRAC-based defenses are non-secure or not practical

(c) QPRAC with Priority Queues enables secure & practical mitigation

Fig. 1. (a) With the PRAC framework, DRAM can request a time for mitigation when it needs it (based on per-row activation counters), using Alerts to service its mitigation queue. (b) Existing PRAC implementations are either insecure (Panopticon [2]) due to the usage of FIFO-based queues or impractical (UPRAC [4]) due to the lack of any queues. (c) We propose QPRAC, using a priority-based service queue (PSQ) for mitigations, which can be cleared on Alerts but also *opportunistically* when another bank requests an All-Bank RFM or *proactively* on REFs. We design QPRAC to be both secure and practical.

in 32Gb DRAM chips with 128K rows per bank, identifying the top-N rows requires activating (52ns) and reading PRAC counters from each row. This causes an impractical overhead, locking all the banks for multiple milliseconds per Alert.

**Our Proposal**: We propose QPRAC, the first secure and practical PRAC implementation adhering to the JEDEC standard to address these concerns. QPRAC uses a low-cost and practical service queue to mitigate the highest activated rows while providing security bounds for ultra-low $T_{RH}$ values.

**1. Priority-Based Service Queue**: QPRAC uses a small Priority-Based Service Queue (PSQ) to track frequently activated rows (see Figure 1(c)). Each PSQ entry maintains a row address and its activation count, using the activation count as the priority. While the PSQ can only track a few rows, it is PRAC-aware, enabling precise tracking of activation counts for all DRAM rows. On activations, the PSQ compares the in-DRAM row activation count with existing PSQ entries. If the activated row has a higher count than any PSQ entry, the entry with the lowest count is replaced with the activated row and its count. Thus, using an N-entry queue, the PSQ maintains the top-N highest activated rows between mitigations. As such, the PSQ avoids the security pitfalls of Panopticon, where the FIFO-based service queue can be bypassed when full.

We analytically show that the security of QPRAC with a PSQ is identical to an ideal PRAC implementation if the PSQ size is at least the number of RFMs per Alert (1, 2, or 4). We craft optimized versions of the wave [4] or feinting attacks [38], incorporating the effects of transitive attack mitigations in PRAC, and show QPRAC is secure up to $T_{RH}$ of 44, 29, and 22 for 1, 2, or 4 RFMs / Alert with an $N_{BO}$ of 1, respectively.

**2. Opportunistic Mitigation**: At ultra-low $T_{RH}$ (sub-100), QPRAC frequently triggers Alerts due to low $N_{BO}$ values. The ABO protocol issues All-Bank RFMs as the current DRAM interface cannot identify the specific bank that initiated the Alert. Consequently, all banks receive RFM commands simultaneously, enabling other banks to *opportunistically* mitigate rows, even if their activation counts are below $N_{BO}$. This avoids future Alerts for these rows and improves performance. For instance, at $N_{BO}$ of 32, QPRAC without opportunistic mitigations incurs a significant 12.4% overhead. Opportuinsitc mitigations considerably reduce this overhead to just 0.8%.

**3. Proactive Mitigation**: DRAM devices receive periodic refresh commands (REF) to ensure charge retention in DRAM cells. Existing DDR4 and DDR5 DRAM currently issue RH mitigations [16], [21], [38] in the shadow of such REF commands. QPRAC can also leverage these mitigations by *proactively* mitigating the highest activated row in the PSQ during a REF. These mitigations provide modest security benefits, reducing $T_{RH}$ by 4 to 10 activations at $T_{RH}$ below 100, and offer performance benefits by lowering Alert frequency. For example, at $N_{BO}$ of 32, QPRAC with proactive mitigations reduces the slowdown from 0.8% to 0%. However, performing proactive mitigation on every REF has high energy overheads, increasing energy consumed by 14.6%. Moreover, not all proactive mitigations are useful, as many entries from the PSQ may not even reach $N_{BO}$. Leveraging this, we propose an energy-optimized design that performs proactive mitigation *only* when the activation count of the highest activated row in the PSQ meets or exceeds a Proactive Mitigation threshold ($N_{PRO}$). This reduces the energy overhead of proactive mitigation from 14.6% to 1.9%, without impacting performance.

Overall, this paper makes five key contributions:

1) We show that existing PRAC implementations are either insecure or impractical.
2) We argue that the service queue design in PRAC is crucial for security, especially with a non-blocking Alert specification, potentially leading to overwhelmed queues.
3) We propose the first practical and secure PRAC implementation, QPRAC, using priority queues.
4) We analyze the security of QPRAC and show that it is secure up to a minimum $T_{RH}$ as low as 22.
5) We realize the potential of *opportunistic* mitigation and co-design QPRAC with a *proactive* mitigation mechanism, providing performance and security benefits.
6) We further enhance our design, including an energy-optimized proactive mitigation scheme, significantly reducing energy overhead while preserving performance benefits.

QPRAC, with an $N_{BO}$ of 32 and one mitigation per Alert, securely handles a $T_{RH}$ of 71. Across 57 workloads, QPRAC incurs 0.8% slowdown, which goes down to 0% with Proactive Mitigation, compared to a non-secure baseline without Alerts. QPRAC requires <15 bytes of storage per DRAM bank.

## II. Background and Motivation

### A. Threat Model

We assume an attacker can issue memory requests for arbitrary rows, knowing the defense algorithm. Our defense aims to prevent single-sided and multi-sided Rowhammer attacks [11], [20] and attacks like Half-Double [31]. The RowPress [35] attack is out of scope since its effects are orthogonal, and it can be mitigated by limiting row open time.

### B. The Rowhammer Vulnerability

Rowhammer (RH) is a read disturbance error where rapid activations of specific rows (aggressors) accelerate charge leakage in neighboring victim rows, leading to bit-flips. The minimum activations required to cause these bit-flips is the RH Threshold ($T_{RH}$). As DRAM technology scales, $T_{RH}$ values have decreased significantly, dropping nearly 16× from 70K in 2014 [29] to 4.5K in 2020 [24], with thresholds likely to drop further in future generations. At ultra-low $T_{RH}$, RH is a bigger security [11], [13], [21], [31], [32], [59] and reliability risk [34]. For future DRAM, it is beneficial to have solutions catering to $T_{RH}$ of 100 or lower.

### C. In-DRAM Rowhammer Mitigation

*1) Commercial Solutions:* Until recently, the DRAM industry relied on two primary approaches to mitigate Rowhammer:

- **Targeted Row Refresh (TRR)**: DRAM manufacturers implemented Targeted Row Refresh (TRR) in DDR4, LPDDR5, and HBM2 to prevent RH [20], [43]. TRR tracks potential aggressor rows using small counter tables or probabilistically and refreshes neighboring victim rows within a blast radius (BR) around the aggressor (e.g., BR = 2 means two victim rows on either side of the aggressor row are refreshed) every few REFs [16], [38]. However, due to limited storage in DRAM, these counter tables have only a few entries, making them vulnerable to malicious patterns that thrash the tables and bypass protections. Several attacks [11], [16], [20], [21], [38] have circumvented TRR and induced RH bit-flips.
- **Refresh Management (RFM)**: As REF-based mitigations do not scale with decreasing $T_{RH}$ values, DDR5 introduced the Refresh Management (RFM) command. This command allows the memory controller to track the total activations issued to a bank. When the number of activations exceeds a specified threshold, the controller issues an RFM command, giving the DRAM time to perform mitigations.

*2) Academic Solutions:* State-of-the-art in-DRAM solutions, such as PrIDE [19] and MINT [47], use probabilistic sampling and FIFO-based or single-entry trackers. These solutions can tolerate a $T_{RH}$ of 1700 with one mitigation per tREFI and a $T_{RH}$ of up to 400 with 4 RFMs per tREFI with negligible slowdown. However, scaling to lower $T_{RH}$ (250 or lower) requires additional RFMs. For example, performing an RFM every 10 activations incurs an activation bandwidth loss of nearly 30%, as each RFM takes 350ns. This results in prohibitive performance overheads.

### D. Per Row Activation Counting (PRAC)

JEDEC's DDR5 specification [40] introduces the Per Row Activation Counting (PRAC) framework to precisely count row activations and holistically mitigate Rowhammer attacks. PRAC consists of two key mechanisms: (1) an activation counter added to each DRAM row, using additional DRAM cells and sense amplifiers, which is incremented on each activation of the corresponding row, and (2) the Alert Back-Off (ABO) protocol, which the DRAM uses to request extra mitigation time when a Rowhammer threat is detected.

The ABO protocol allows DRAM to assert the Alert signal when a row's activation counter crosses a Back-Off threshold ($N_{BO}$). The memory controller can then issue activations for only 180ns ($ABO_{ACT}$ activations) before sending a pre-configured number of RFMs ($N_{mit}$) to allow DRAM to perform RH mitigations to victim rows. After these mitigations, the next Alert can be asserted by the ABO protocol *only* after the DRAM services a pre-specified number of row activations ($ABO_{Delay}$). Additionally, PRAC requires updated DRAM timings to account for the time needed to increment the in-DRAM activation counters. Table II showcases the PRAC-related parameters, their explanation, and values for our evaluation.

TABLE I
PRAC Parameters as per DDR5 specification [40]

| Parameter | Explanation | Value |
|---|---|---|
| $N_{BO}$ | Back-Off Threshold | $N_{BO} \leq T_{RH}$ |
| $N_{mit}$ | Num RFMs on Alert | 1, 2, or 4 |
| $ABO_{ACT}$ | Max. ACTs from Alert to RFM | 3 (up to 180ns) |
| $ABO_{Delay}$ | Min. ACTs after RFM to Alert | Same as $N_{mit}$ (1,2, or 4) |

### E. Existing PRAC Implementations and their Drawbacks

While the DDR5 specification proposes the PRAC interface, its implementation is left to the DRAM vendors, making the security guarantees of such solutions unclear. Below, we describe two PRAC implementations and their drawbacks.

*1) Insecurity of Panopticon:* Panopticon [2] inspired the PRAC design by proposing activation counters for each DRAM row. When a counter crosses the mitigation threshold, a threshold bit $t$ toggles, identifying the row for mitigation. A *FIFO-based* service queue tracks these rows. If the queue is full, the DRAM uses the ABO protocol to halt activations and request RFMs for RH mitigation, freeing up queue entries. Additionally, REF commands can mitigate rows from the service queue, which further frees up entries.

**Vulnerability**: Panopticon is insecure when implemented with the PRAC specification because of three reasons: (1) mitigating rows only upon toggling of the counter's threshold bit (t-bit), (2) a limited capacity of FIFO-based service queue, and (3) PRAC's non-blocking nature of Alerts.[1] These issues render a PRAC implementation based on Panopticon insecure. We elaborate on these with two illustrative attacks.

[1]While these vulnerabilities could be avoided by making Alerts immediately blocking, this would violate JEDEC's PRAC specifications, which allow up to 180ns for servicing Alerts. This delay is necessary for the memory controller to receive the Alert from DRAM, read the Mode Register, and confirm whether it was triggered by the ABO protocol or other issues, such as CRC errors, before initiating the ABO protocol for RH mitigations [40].

**(1) *Toggle+Forget Attack* – Exploiting t-bit Toggling:** Since PRAC uses a non-blocking Alert, the memory controller can issue up to $ABO_{ACT}$ activations even after receiving an Alert. These activations can be exploited when the service queue is full, enabling a row to bypass the queue if its $t$-bit is toggled by the $ABO_{ACT}$. After a bypass, subsequent activations to that row will not trigger immediate RH mitigation since the $t$-bit does not toggle again until other $2^t$ activations. This allows a row to be activated indefinitely without mitigation until the end of tREFW (32ms). This attack is outlined below.

Assuming a service queue size of Q and a mitigation threshold of M, our attack activates Q+1 rows, starting from the same activation count. The rows are activated uniformly in a round-robin manner. After each row receives M-1 activations, the next activation of the first Q rows pushes each of them into the service queue, filling it, and causing the DRAM to raise the Alert. While the queue is full, using the $ABO_{ACT}$, the Q+1th row (target row) is activated twice, allowing it to escape insertion into the queue and mitigation. After the Alert is serviced with the RFMs and the queue is no longer full, the Q rows are activated twice to bring them to the target row activations. Then, the previous attack step is repeated on Q+1 rows [2]. This process continues until the tREFW period (32ms).

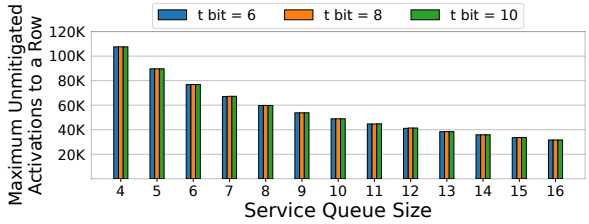



Fig. 2. The security vulnerability of Panopticon [2] due to t-bit toggling, i.e., maximum activations before the row receives a mitigation with *Toggle+Forget Attack*. For sub-100 $T_{RH}$, our attack can cause a DRAM row to receive even $100\times T_{RH}$ activations without any mitigation. This vulnerability is independent of the mitigation threshold ($2^t$) used by Panopticon.

Figure 2 shows the maximum activations to the target row in the above attack before mitigation, with queue sizes varying from 4 to 16 for different mitigation thresholds of 32, 512, or 1024 (t-bit of 6, 8, or 10 respectively). The target row receives no mitigation until the end of tREFW. Consequently, it can be activated beyond 100K times without mitigation with a queue size of 4 and about 25K times with a queue size of 16. As the queue size increases, the maximum activations decrease linearly as the pool of rows to be uniformly activated increases. No matter the queue size, the target row is activated beyond $100\times$ of $T_{RH}$ (for $T_{RH}$ of sub-100), compromising the security of Panopticon. Moreover, this behavior is independent of the mitigation threshold (t-bit) used, indicating that reducing the mitigation threshold cannot address the vulnerability.

One way to address this vulnerability is to extend Panopticon by using larger counters that never overflow and comparing the full counter value against the threshold to identify rows to be mitigated. Thus, even if $ABO_{ACT}$ causes a counter to cross the threshold and temporarily skip mitigation due to the queue being full, the row would be inserted into the queue on subsequent ACTs. However, this design is still insecure.

**(2) *Fill+Escape Attack* – Exploiting FIFO Service Queues:** Assuming the full counter-value is compared with the threshold on each activation (so t-bit toggling cannot be exploited), an attacker can still avoid mitigations for a target row by hammering it *only* with $ABO_{ACT}$ *and* only when the FIFO-based service queue is full.

In this attack, the attacker first activates the target row and Q other rows, with M-1 activations each. The attacker then activates Q rows by one activation to make the FIFO queue full. When the Alert is raised, the three $ABO_{ACT}$ activations to a target row can raise the activation count without the row being selected for mitigation. After the Alert is serviced, up to four entries will be removed from the queue (and one extra entry may be removed due to mitigation on tREFI). The attacker will again fill the queue by activating five other rows to M activations. Thus, with every $5 \times M$ additional ACTs, the target row receives three extra activations (via $ABO_{ACT}$) without a mitigation.

Figure 3 shows the maximum unmitigated activations to the target row using this attack, as the mitigation threshold (M) ranges from 64 to 4096. The attacker can achieve a minimum of 1283 unmitigated ACTs on the target row at a mitigation threshold of 512. In fact, at lower mitigation thresholds, the number of unmitigated activations to target row increases dramatically, as filling up the FIFO queues requires less effort. Thus, even the optimized version of Panopticon is insecure below a $T_{RH}$ of 1280, primarily due to filled FIFO service queues combined with non-blocking Alerts in PRAC, allowing a high number of unmitigated activations.

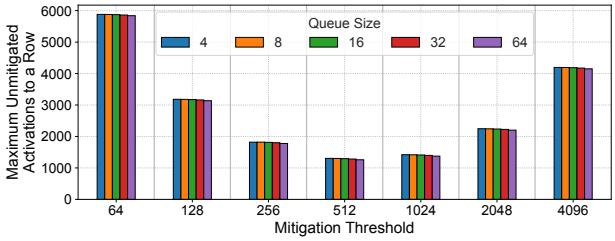



Fig. 3. The security vulnerability of Panopticon (with full counter comparisons) under the *Fill+Escape Attack*, which exploits filled FIFO-based service queues. Combined with non-blocking Alert, this allows at least 1283 unmitigated ACTs (at a mitigation threshold of 512), with the number increasing at lower thresholds.

*2) Impracticality of UPRAC:* UPRAC [4] proposes a PRAC implementation *without* a service queue. It raises an Alert when any DRAM row crosses the Back-Off threshold ($N_{BO}$) and proposes to mitigate the N highest activated rows globally.

**Impractical Overhead of UPRAC:** While this design does not use a service queue and avoids related security vulnerabilities, it is impractical. Without a service queue, on an Alert, it is impractical for the DRAM to read the activation counters of *all* the DRAM rows to identify the top-N rows.

[2]While Panopticon [2] can initialize rows with random activation counts, the attacker can easily get rows to a known activation count. The paper [2] shows that activating randomly chosen rows can cause the queues to be full and Alert raised in tens of minutes, bringing the activation count of the last row to a known value (a multiple of the threshold). This can be repeated Q+1 times, to get Q+1 rows with activation counts at a multiple of the threshold.

**Vulnerability of UPRAC + FIFO Service Queues:** UPRAC can be practical using a FIFO-based service queue, requiring mitigation when the row activation count exceeds a threshold lower than $N_{BO}$. However, this approach is also vulnerable to the *Fill+Escape Attack* on Panopticon (Section II-E1), which exploit full FIFO queues in combination with non-blocking Alerts. Entries can be inserted at a maximum rate of one per activation, whereas removal occurs at best at one per four activations ($ABO_{ACT}$ + $ABO_{Delay}$). An attacker can thus fill up the UPRAC FIFO queue with Q rows, each activated to $N_{BO}$, and then use three $ABO_{ACT}$ to hammer the target row each time. This incurs at least 1283 ACTs to a target row without mitigation (at $N_{BO}$ of 512) and higher ACTs at lower $N_{BO}$. Thus, UPRAC is insecure below the $T_{RH}$ of 1280.

**Key Question:** Designs without service queues are impractical, and practical designs using FIFO queues are insecure at sub-1000 $T_{RH}$. Can we design a PRAC implementation that is both practical and secure at ultra-low $T_{RH}$ (sub-100)?

### F. Goal

Our goal is to design a secure PRAC implementation that provides strong security guarantees at ultra-low $T_{RH}$ (sub-100). At the same time, we seek a practical service queue design that avoids the insecurity of FIFO-based queues without modifying the JEDEC PRAC specification to ensure that DRAM vendors can easily adopt it. To that end, we explore a priority-based service queue design using activation counts to ensure secure and scalable Rowhammer mitigation.

## III. Design of QPRAC

This paper proposes QPRAC, a PRAC-based implementation to enable practical and secure Rowhammer mitigation. The key focus of our solution is the *Priority-based Service Queue* (PSQ), which provides a practical mechanism to track pending mitigation without the security vulnerabilities introduced by FIFO-based service queues. Below, we provide an overview of QPRAC's design, its queue management policies (insertion, eviction, and mitigation), and finally, its co-design with other mitigation opportunities available to DRAM.

### A. Overview of QPRAC

Any implementation of the PRAC specification needs to answer the following questions: (1) how to select a row for RH mitigation, (2) how to track rows identified for RH mitigation while it is pending, and (3) when to request the memory controller for additional time for RH mitigation.

As shown in Figure 4, QPRAC addresses these using three key components: (1) per-row activation counters in DRAM, as specified by PRAC specification [40], (2) a per DRAM bank priority-based service queue (PSQ) that tracks the highest activated rows even in situations where the queue is full, and (3) an implementation of the Alert Back-Off (ABO) protocol to mitigate rows identified in the PSQ in a timely manner. We now explain these components.

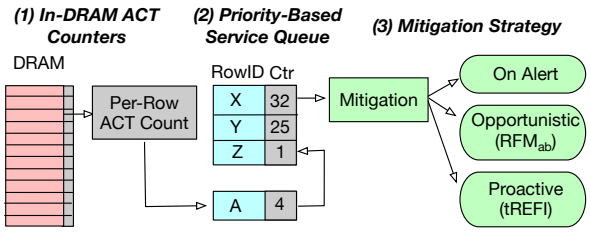



Fig. 4. Overview of QPRAC design. It consists of three components: (1) PRAC-based in-DRAM activation counters, (2) a Priority-Based Service Queue (PSQ) to identify rows to be mitigated, and (3) a strategy that uses Alert-based, opportunistic, and proactive RH mitigations.

### B. Priority-Based Service Queue Design and Operation

*1) Design:* QPRAC consists of an N-entry priority-based service queue (PSQ) per DRAM bank. As shown in Figure 5, each entry represents a row and includes its RowID and current activation count. The queue is sorted in descending order by activation count, prioritizing rows with higher activation counts. We design the PSQ using a CAM (content-addressable memory); we assume small PSQs with five entries per bank.

*2) Operation:* When a row is activated, its in-DRAM activation counter is incremented according to the PRAC specification. Simultaneously, the activated row is also considered for insertion into the service queue. The PSQ inserts only rows with activation counts higher than the lowest count in the queue. If a new entry is inserted, the entry with the lowest count is evicted. If the activated row is already in the queue, its activation count is updated to match the in-DRAM count.

*3) Intuition:* Unlike prior works, which are vulnerable when the service queue becomes full, the PSQ is *intentionally* designed to be full at all times. This design ensures that the PSQ *always* retains and tracks the highest activated rows. Consequently, the PSQ cannot lose information about heavily activated rows, even in attack scenarios that activate more rows than the service queue's capacity – a scenario that compromises the security of previous solutions.

### C. Mitigation Policy using Alert Back-Off Protocol

*1) Design:* QPRAC uses the ABO protocol to request DRAM mitigations. Unlike prior defenses that use different thresholds for issuing mitigation and signaling an Alert, QPRAC simplifies this process. It tracks rows with the highest activation counts in the PSQ, using a single threshold to flag the highest priority row for mitigation and raise an Alert.

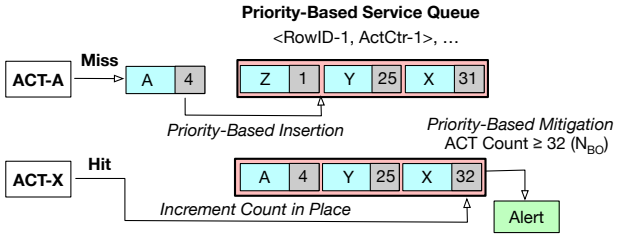



Fig. 5. Design of Priority-Based Service Queue (PSQ). Any activation can insert a row into PSQ based on priority (activation count) on misses and increment count on hits. PSQ raises an Alert if any count is at $N_{BO}$ or above.

*2) Operation:* When the activation count of the highest activated row in the PSQ crosses the Back-Off threshold ($N_{BO}$), QPRAC identifies the need for an RH mitigation. It asserts the Alert signal to the memory controller to initiate the ABO process and issue a mitigation. The memory controller then issues $N_{mit}$ RFMs (default $N_{mit}$ is 1) to allow the DRAM to mitigate $N_{mit}$ rows with the highest counts in the PSQ. For each RFM, the DRAM mitigates the aggressor with the highest activation count in the PSQ, refreshing the blast-radius (BR) victim rows above and below it (default BR is 2). Additionally, the aggressor's in-DRAM per-row counter is reset to 0 by activating it, and its entry is evicted from the PSQ.

To mitigate transitive attacks like Half-Double, each mitigative refresh to the victim row also increments the in-DRAM counter associated with the victim. The victim row itself may be inserted into the PSQ if its activation count is higher than the minimum activation count of entries in the PSQ.

After the RFMs are serviced, the Alert is de-asserted until $ABO_{Delay}$ activations. The next Alert can be raised if there are rows in the PSQ with activation counts at or beyond $N_{BO}$.

*3) Security:* Any row that crosses $N_{BO}$ is eligible for RH mitigation by triggering an Alert. However, the non-blocking nature of the Alert allows some activations to occur despite the raised Alert, and there is a limit on how frequently Alerts can be raised. This permits a certain number of activations to rows beyond the $N_{BO}$ value. In Section IV, we establish an upper bound on these activations to determine the appropriate $N_{BO}$ value for security at a given $T_{RH}$. Our analysis in Section IV-B shows that as long as the PSQ size matches $N_{mit}$ (the number of mitigations per Alert), QPRAC provides deterministic security against RH attacks, even at sub-100 $T_{RH}$.

### D. Additional Mitigation Opportunities using PSQ

Thus far, we have considered RH mitigation for a row in the PSQ only when its activation count crosses the Back-Off threshold ($N_{BO}$), triggering an Alert. However, there are additional opportunities to provide RH mitigation to entries in the PSQ at no extra performance cost. These additional RH mitigations can reduce the number of future Alerts and improve overall performance.

*1) Opportunistic Mitigation on All-Bank RFM:* When an Alert is raised for a row that crosses the $N_{BO}$, the memory controller must send an All-Bank RFM ($RFM_{ab}$) command(s) to all DRAM banks. This is necessary because the current DRAM Alert interface cannot specify which bank issued the Alert. Consequently, the memory controller must stall all banks with the $RFM_{ab}$ command.

An $RFM_{ab}$ allows for opportunistic mitigations for PSQ entries in other banks, even if their activation counts are below the $N_{BO}$. QPRAC takes advantage of this by issuing opportunistic mitigations to the $N_{mit}$ highest activated rows in all banks, regardless of their activation count. This approach mitigates rows across all banks before they reach the $N_{BO}$ and more importantly, reduces the number of future Alerts and the overall slowdown due to the mitigations.

*2) Proactive Mitigation on REF commands:* Similar to TRR in DDR4, which mitigates aggressor rows during REF, PRAC can benefit from proactive mitigations issued during REF operations [2], [4]. Proactive mitigations, like opportunistic mitigations, target the row with the highest activation count in the PSQ of each DRAM bank, regardless of whether its count exceeds $N_{BO}$. To support this, the PSQ must be at least $N_{mit}$ + 1 in size to handle mitigations on an Alert ($N_{mit}$) and accommodate an additional entry during a REF command. Unlike opportunistic mitigations, performing proactive mitigations for every mitigation opportunity can incur excessive energy overhead due to their higher frequency compared to Alerts. To address this, we propose an energy-optimized approach that performs mitigations *only* when the activation counter of the highest activated row in the PSQ of each bank meets or exceeds the Proactive Mitigation threshold ($N_{PRO} = \frac{N_{BO}}{K}$), significantly reducing energy consumption while maintaining performance.

### E. Sizing the Structures

**Sizing PSQ**: A PSQ size of $N_{mit}$ + 1 is essential for the security of QPRAC with proactive mitigation. This size ensures that QPRAC can properly handle mitigations for an Alert ($N_{mit}$) while also accommodating additional mitigation during refresh. Since the PRAC specification supports $N_{mit}$ values of 1, 2, and 4 [40], we use a PSQ size of 5 for QPRAC. **Sizing Counters**: Our counters are sized to avoid overflows. As per the bounds for our maximum activation count in Figure 13, we set their size as minimum$(6,\ \log_2(T_{RH})+1)$ bits. In practice, we use 7-bit counters for a $T_{RH}$ of 66.

## IV. Security Analysis

We determine the Rowhammer threshold ($T_{RH}$) that QPRAC can securely defend against by analyzing worst-case attack patterns at different Back-Off thresholds ($N_{BO}$). We perform this analysis for an idealized PRAC implementation, which assumes that top-N highest activated rows are mitigated on each Alert. We then extend this to QPRAC, which uses PSQs.

### A. Analyzing Security of an Ideal PRAC Implementation

To model worst-case attacks on PRAC, we assume:

- **Mitigation Only via Alert Back-Off:** As the JEDEC specification does not specify the policy for mitigations on REFs, we bound the security of PRAC without assuming any mitigation on REFs. Thus, this results in a *pessimistic* upper bound on the Rowhammer threshold.
- **Each Alert Mitigates Top-N Activated Rows:** This models an Idealized PRAC, where each Alert mitigates the globally top-N activated rows in the bank, using 'N' RFMs. We show how the security guarantees for QPRAC are similar to PRAC-Ideal in Section IV-B.

*1) Modeling Wave or Feinting Attack on PRAC:* Similar to prior work [4], we model the Wave or the Feinting attack [38]. This state-of-the-art attack maximizes row activations before mitigation in a PRAC-protected DRAM. This multi-round attack starts with a pool of rows, activating each row once per round. In each round, the attacker identifies and drops the mitigated rows from the pool, then uniformly activates the remaining rows. In the final round, where all remaining rows will be mitigated at the next instance, the attack focuses on hammering a single row.

The attack on PRAC consists of two phases: the *Setup* phase and the *Online* phase. In the Setup phase, a pool of rows of size $R_1$ is generated, with each row activated $N_{BO}-1$ times to avoid mitigation. In the Online phase, the attack proceeds through multiple rounds, uniformly activating the remaining rows in each round until only a single row remains in the final round, which then receives focused hammering. Assuming the row that lasts until the final round receives a maximum of $N_{online}$ activations, the $T_{RH}$ at which the defense is secure is:

$$T_{RH} > N_{BO} + N_{online} \quad (1)$$

*2) Bounding the Online Phase Activations:* To bound the Online Phase activations ($N_{online}$), we consider the number of attack rounds (NR), with one activation per round, followed by $ABO_{Delay}$ + $ABO_{ACT}$ activations, which are possible before the last Alert in the final round. Additionally, the penultimate Alert's RH mitigations can cause blast-radius (BR) activations if the row in the last round is a neighbor of the mitigated rows. This increases the activations to the last row by BR.

$$N_{online} = NR + ABO_{ACT} + ABO_{Delay} + BR \quad (2)$$

The number of rounds (NR) can be derived by assuming we start with a pool of $R_1$ rows and recursively calculating the pool of rows ($R_N$) at each round N. In each round, the pool reduces by the number of mitigated rows ($N_{mit}$) $\times$ Number-of-Alerts. Since the RFMs of the last Alert in a round provide BR activations for free, each round only activates $R_{N-1}$ - BR rows, as an Alert occurs every $ABO_{ACT}$ + $ABO_{Delay}$ activations. This allows us to determine each round's pool of rows ($R_N$).

$$R_N = R_{N-1} - \lfloor N_{mit} * (R_{N-1} - BR) / (ABO_{ACT} + ABO_{Delay}) \rfloor \quad (3)$$

Using Equation (3), we can recursively calculate the total number of rounds (NR), given $R_1$, and then use Equation (2) to determine $N_{online}$ as $R_1$ varies. As shown in Figure 6, $N_{online}$ increases with the starting row pool size. With a maximum of 128K rows (total rows in the bank), $N_{online}$ can reach 46 for PRAC-1 ($N_{mit}$ = 1), 30 for PRAC-2 ($N_{mit}$ = 2), and 23 for PRAC-4 ($N_{mit}$ = 4).

*3) Constraint on $R_1$ Due to Attack Time:* The starting pool size of the attack ($R_1$) is constrained by the time required for both the Setup and Online phases. The Setup phase involves activating $R_1$ rows $N_{BO}$-1 times each, and both phases must be completed within tREFW (32ms) for the attack to succeed, limiting $R_1$. Figure 7 shows the maximum $R_1$ as $N_{BO}$ varies. At $N_{BO}$ of 1, the Setup phase requires negligible time, and $R_1$ is limited by the Online phase duration, ranging from 50K to 62K for PRAC-1 to PRAC-4. As $N_{BO}$ increases to 256, the maximum $R_1$ size drops to 2K due to the Setup phase dominating. As we move from PRAC-1 to PRAC-4, the time required for the Online phase decreases for a given $R_1$ since the number of mitigations per Alert increases. Consequently, for the same $N_{BO}$, $R_1$ increases from PRAC-1 to PRAC-4.

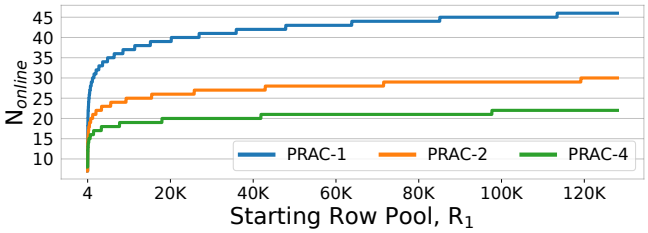



Fig. 6. Maximum Row Activations in Online Attack ($N_{online}$) versus Starting Row Pool Size ($R_1$) using an analytical model. $N_{online}$ reaches a maximum of 46, 30, and 23 for PRAC-1, PRAC-2, and PRAC-4. In empirical evaluations of the attack, our results were within 1% of the analytical results.

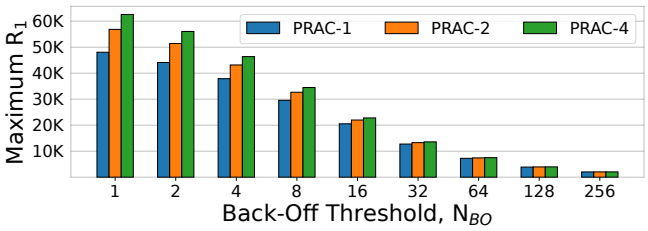



Fig. 7. Maximum Starting Row Pool ($R_1$) versus Back-Off Threshold ($N_{BO}$). As $N_{BO}$ increases, the maximum possible $R_1$ decreases. This is because the time taken by the setup phase increases at higher $N_{BO}$.

*4) Quantifying $T_{RH}$:* Using the constraints on $R_1$ for different $N_{BO}$ from Figure 7 and Figure 6, we can determine the maximum $N_{online}$ value and subsequently the $T_{RH}$ that PRAC can tolerate, using Equation (1). Figure 8 shows the lowest possible $T_{RH}$ for which PRAC is secure at different $N_{BO}$s. At $N_{BO}$ of 1, PRAC-1, PRAC-2, and PRAC-4 are secure for $T_{RH}$ values of 44, 29, and 22, respectively. As $N_{BO}$ increases, the value of $N_{online}$ remains relatively unchanged. At $N_{BO}$ of 256, the securely mitigated $T_{RH}$ values are 289, 279, and 274 for PRAC-1, PRAC-2, and PRAC-4, respectively.

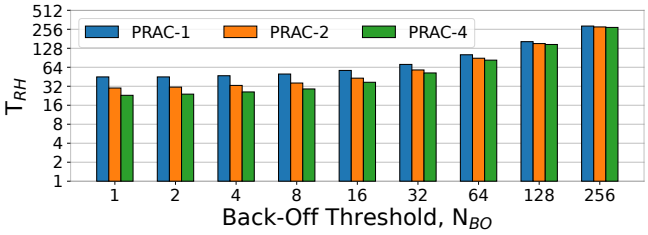



Fig. 8. $T_{RH}$ values for which PRAC-N is secure as $N_{BO}$ varies. At $N_{BO}$ of 1, the lowest possible $T_{RH}$ for PRAC-1, 2, and 4 is 44, 29, and 22, respectively.

A similar analysis in prior work, UPRAC [4], fails to account for the activations in the last round for a single row and the effect of transitive attack mitigation. This can exacerbate the attack and the Setup phase time, which bounds the pool of rows in the attack. Unlike prior claims [4] that PRAC-1 to PRAC-4 are secure at a minimum $T_{RH}$ of 17 to 10, our precise modeling shows that they are secure only up to $T_{RH}$ of 44 to 22, respectively.

### B. Effect of Priority-Based Service Queue in QPRAC

Unlike an 'Ideal' PRAC, QPRAC uses a priority-based service queue (PSQ) of $N$ entries ($N \geq N_{mit}$) to determine the top $N_{mit}$ rows to mitigate when an Alert is raised. So, we seek to answer two questions: (1) Is the PSQ secure from attacks that exploit full FIFO queues? (2) Is the PSQ vulnerable to any new attack patterns due to limited queue capacity?

**Tolerating Full PSQ**: The PSQ design addresses the security limitations of FIFO-based queues when they are full. While FIFO is vulnerable to attacks (Section II-E1-*Fill+Escape Attack*), where highly activated rows hammered with $ABO_{ACT}$ are bypassed when the queue is full, as shown in Figure 9, priority-based insertions in QPRAC insert such rows even when PSQ is full. This effectively tracks and mitigates such rows, making QPRAC secure from such attacks.

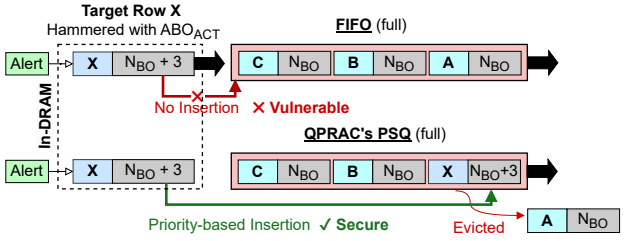



Fig. 9. FIFO-based queues vs QPRAC's PSQ. FIFO is vulnerable to insertion bypass when full, while PSQ uses priority-based insertion, prioritizing rows with higher activation counts to secure against $ABO_{ACT}$-based hammering.

**Security Holds Under Size Constraint**: A size-constrained PSQ, however, means it cannot always hold the globally top $N$ rows. Consequently, mitigations can target rows with the highest global activation counts or local maxima. Yet, under the wave attack discussed in Section IV-A, all mitigation decisions executed through the PSQ consistently target the globally most frequently activated rows, thus aligning with the 'Ideal' PRAC under this attack.

This is because the wave attack uniformly activates pool rows to the same maximum activation count. Even if some rows are evicted from the PSQ due to insufficient capacity, as demonstrated in Figure 10(a), they are reinserted into the PSQ the next time they are activated. Therefore, during attacks, including but not limited to the wave attack, the PSQ predominantly holds the top-$N$ most frequently activated rows. Simulations of the Wave or Feinting attack show that the maximum activation counts for QPRAC (with PSQ) are identical to those of the ideal PRAC (without PSQ).

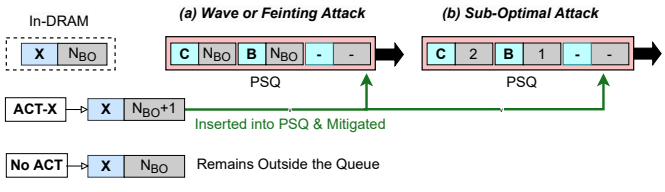



Fig. 10. QPRAC's tracking of global maximums. Sometimes, the global maximum may be outside the PSQ, but it cannot be activated further without being inserted into the queue and mitigated.

**Alternative Attacks Are Inferior**: Any attack variant that forces mitigations on locally (not globally) maximum activation counts is sub-optimal. This happens only if (1) the global maximum is evicted from the PSQ and not activated again, and (2) other rows with fewer activation counts are activated, inserted into the PSQ, and mitigated on Alert. Such an attack is sub-optimal because, as shown in Figure 10(b), the globally maximum activated row cannot increase its activation outside the PSQ. Time spent keeping the global maximum row outside of the PSQ is wasted. When activated again, the global maximum row is reinserted into the PSQ and preserved until mitigation. Thus, the PSQ *does not* introduce worse attacks.

### C. Effect of Proactive Mitigation on QPRAC

QPRAC can be co-designed with proactive mitigation on REFs, where one mitigation per REF is proactively issued for the highest activated row in the PSQ, as long as the queue size is at least $N_{mit}+1$. This approach has two key benefits: (1) reducing the number of rows that can reach $N_{BO}$ in the *Setup* phase, and (2) mitigating more rows per round in the *Online* phase, thereby reducing the number of attack rounds.

*1) Impact on Setup Phase:* Rows are uniformly activated to reach $N_{BO}$ - 1 activations during the Setup phase. Proactive mitigations reduce the pool of rows ($R_1$) available for the attack. The number of mitigations ($M$) is calculated as the number of activations ($A$) in the Setup phase divided by the number of activations per tREFI (67), as $M = \frac{A}{67}$. As shown in Figure 11, with proactive mitigation, the pool of rows ($R_1$) available for the attack decreases. For $N_{BO}$ values of 16 and higher, proactive mitigation significantly reduces $R_1$ compared to PRAC alone, with $N_{BO}$ values of 128 and 256 completely defeating the attack by mitigating the entire $R_1$ pool before it crosses $N_{BO}$. For $N_{BO}$ lower than 16, the short Setup phase does not experience as many mitigations; the online phase being shorter with the proactive mitigations allows for larger values of $R_1$ to be possible for the attack.

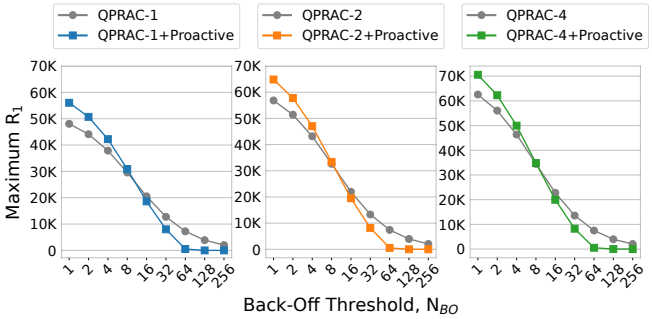



Fig. 11. Maximum Starting Row Pool Size ($R_1$) in the wave attack for QPRAC with proactive mitigation compared to QPRAC without proactive mitigation (labeled simply as QPRAC). For higher $N_{BO}$, where the Setup phase consumes more time, the $R_1$ size reduces considerably due to proactive mitigation.

*2) Impact on Online Phase:* In the Online Phase, the pool of rows in each round decreases more rapidly due to proactive mitigations. The number of additional rows mitigated is calculated by dividing the total Onlie phase time (Activation-Time + Alert-Time) by tREFI. Including these in Equation (3), we calculate $N_{online}$ versus $R_1$, shown in Figure 12.

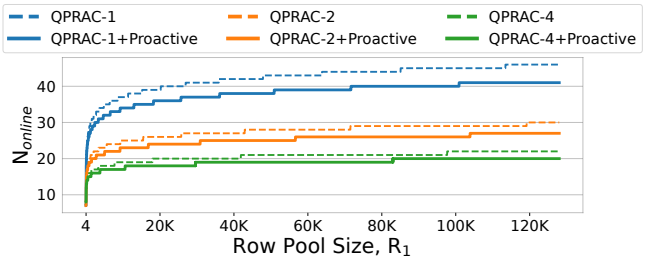



Fig. 12. Maximum Activations Per Row in Online Phase ($N_{online}$) for QPRAC *with* proactive mitigation versus QPRAC *without* proactive mitigation (labeled simply as QPRAC). $N_{online}$ decreases by a maximum of 5, 2, and 1 for QPRAC-1, QPRAC-2, and QPRAC-4 with proactive mitigations, respectively.

Using $R_1$ from Figure 11 and the associated $N_{online}$ from Figure 12, we can determine the minimum $T_{RH}$ supported by QPRAC with proactive mitigations, as shown in Figure 13. For $N_{BO}$ of 1, the minimum supported $T_{RH}$ drops to 40, 27, and 20 for PRAC-1, 2, and 4, respectively, with proactive mitigation, compared to 44, 29, and 22 without proactive mitigation. For our default $N_{BO}$ of 32, proactive mitigation can defend against a $T_{RH}$ of 66, 55, and 50 for QPRAC-1, 2, and 4, respectively, compared to 71, 58, and 52 without proactive mitigation.

Our energy-aware design, QPRAC with energy-aware proactive mitigations (QPRAC+Proactive-EA), which skips some wasteful proactive mitigations, achieves a security level between QPRAC and QPRAC+Proactive. This is because it also reduces the number of rows available in the *setup* phase but to a lesser extent than proactive mitigation on every REF.

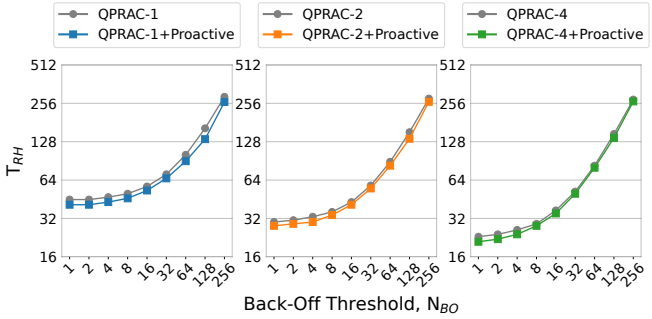



Fig. 13. The $T_{RH}$ values for QPRAC with proactive mitigation and without proactive mitigation (labeled simply as QPRAC). With proactive mitigation, the lowest possible $T_{RH}$ at $N_{BO}$ of 1 is 40, 27, and 20 for QPRAC-1, 2, and 4, respectively. In contrast, the lowest possible $T_{RH}$ without proactive mitigation is 44, 29, and 22 for QPRAC-1, 2, and 4, respectively.

## V. Evaluation Methodology

**Simulation Framework**: We evaluate designs using the cycle-accurate trace-based DRAM simulator Ramulator2 [30], [36]. We use an out-of-order core model in Ramulator2, similar to prior RH works [3], [4], [35], [44], [63], [65], [66]. Our system configuration is shown in Table II. We simulate a baseline system with a 4-core, 8MB shared LLC equipped with 64GB DDR5 memory (one channel, two ranks). The memory is configured using timing parameters based on the Micron 32Gb DDR5 device [41], including PRAC-specific timing changes [40]. Each DRAM bank consists of 128K rows, each 8KB in size. Within a 32ms refresh window, a single bank can undergo up to approximately 550K activations.

TABLE II
System Configuration

| | |
|---|---|
| Out-Of-Order Cores | 4 Core, 4GHz, 4 wide, 352 entry ROB |
| Last Level Cache (Shared) | 8MB, 8-Way, 64B lines |
| Memory Size, Type | 64 GB, DDR5 |
| Bus Speed | 3200MHz (6400MHz DDR) |
| DRAM Organization | 4 Bank x 8 Groups x 2 Ranks x 1 Channel |
| tRCD, tCL, tRAS | 16ns, 16ns, 16ns |
| tRP, tRTP, tWR, tRC | 36ns, 5ns, 10ns, 52ns |
| tRFC, tREFI | 410 ns, 3.9$\mu$s |
| $tABO_{ACT}$, $tRFM_{ab}$ | 180ns, 350ns |
| Rows Per Bank, Size | 128K, 8KB |

**Evaluated Designs**: We compare QPRAC against a baseline DRAM that also uses DDR5 PRAC timings but without the Alert Back-Off (ABO) based mitigations. We extend Ramulator2 to faithfully model the per-row activation counters, the ABO protocol, and their timing constraints. We evaluate the following QPRAC configurations: 1) **QPRAC-NoOp** that performs mitigation on an RFM *only* for the bank with the entry that reached the Back-Off Threshold ($N_{BO}$). 2) **QPRAC** that mitigates the highest activated row(s) in the priority-based service queue (PSQ) from every bank when an RFM is received. 3) **QPRAC+Proactive** that additionally performs proactive mitigation for the highest activated entry during the refresh operations (REF) for each bank. 4) **QPRAC+Proactive-EA** is an energy-aware extension of QPRAC+Proactive. It mitigates the most frequently activated rows in the PSQ during proactive mitigations only when their counter reaches $N_{PRO}$. By default, we set $N_{PRO}$ to half of $N_{BO}$. 5) **QPRAC-Ideal** is an ideal implementation that knows and mitigates the 'top-N' highest activated rows for each ABO in addition to proactive mitigations, similar to UPRAC [4].

**Workloads**: We use 57 applications from SPEC2006 [8], SPEC2017 [58], TPC [60], Hadoop [10], MediaBench [12], and YCSB [7] benchmarks, open-sourced with Ramulator2 [49]. We run four homogeneous workloads until each core completes 500 million instructions. We use a default Back-Off Threshold ($N_{BO}$) of 32 and 1 RFM per Alert (PRAC-1). We also vary $N_{BO}$ (128 to 16), the number of RFMs per Alert (1, 2, or 4), and queue sizes (1 to 5). We use the weighted speedup to evaluate the performance of QPRAC designs.

## VI. Results and Analysis

### A. Performance Overhead

Figure 14 shows the performance overhead of QPRAC implementations normalized to a non-secure baseline DDR5 system. QPRAC-NoOp incurs a 12.4% slowdown, while QPRAC, QPRAC+Proactive, QPRAC+Proactive-EA (our default), and QPRAC-Ideal experience minimal overheads of 0.8%, and no overhead for the latter three with proactive mitigations.

The higher overhead for QPRAC-NoOp arises because it only mitigates DRAM banks that reach the Back-Off threshold ($N_{BO}$) when these banks receive the RFMs caused by Alert Back-Off (ABO). Consequently, when other banks reach the $N_{BO}$, they issue separate Alerts followed by RFMs, reducing effective DRAM bandwidth. Opportunistically mitigating all PSQs on each RFM, as implemented in QPRAC by default, significantly reduces the number of Alerts. QPRAC+Proactive

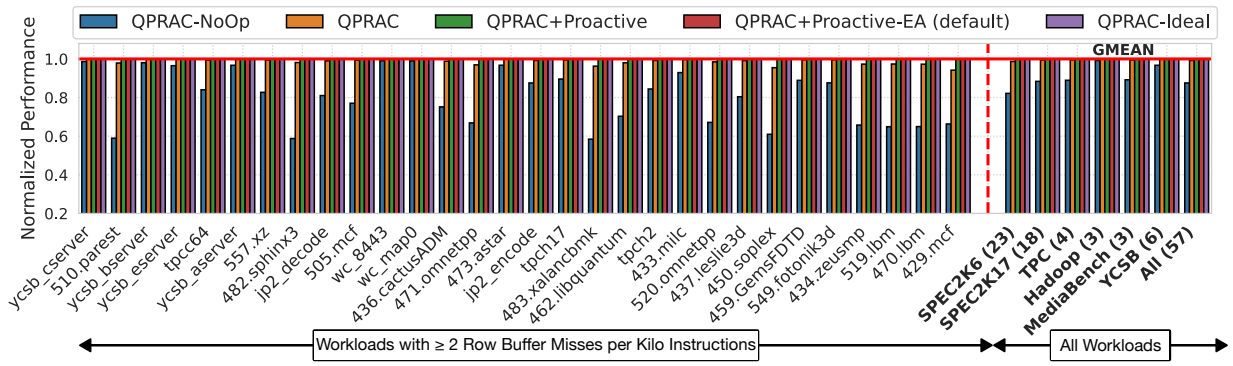



Fig. 14. Normalized performance of QPRAC with a 5-entry priority queue at a Back-Off threshold ($N_{BO}$) of 32 and 1 RFM per Alert Back-Off (ABO), compared to an insecure baseline without ABO. QPRAC-NoOp incurs a considerable 12.4% slowdown on average. In contrast, other QPRAC implementations with opportunistic mitigations–QPRAC, QPRAC+Proactive, QPRAC+Proactive-EA (our default), and QPRAC-Ideal–result in negligible 0.8% slowdown for QPRAC, and no slowdown for the latter three due to proactive mitigations.

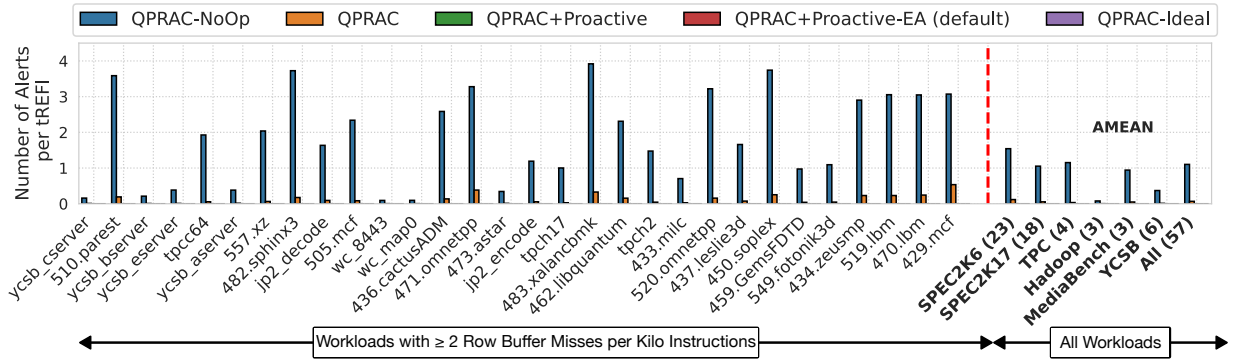



Fig. 15. The frequency of Alert Back-Off (ABO) occurrences per tREFI interval for different QPRAC implementations at a Back-Off threshold of 32 and 1 RFM per ABO. QPRAC-NoOp experiences nearly 1.1 ABO per tREFI. In contrast, QPRAC (with opportunistic mitigation), QPRAC+Proactive, QPRAC+Proactive-EA (our default), and QPRAC-Ideal have insignificant 0.07 ABO per tREFI, with no ABO occurrences for the latter three due to proactive mitigations.

further improves performance by leveraging proactive mitigations to reduce the burden on Alerts, bringing no additional performance overhead. QPRAC+Proactive-EA maintains optimal performance as its *proactive mitigation threshold* ($N_{PRO}$) still ensures aggressor rows are mitigated well before they reach $N_{BO}$, thus ensuring minimal Alerts and avoiding their slowdown. QPRAC-Ideal (which similarly has proactive mitigation) shows identical performance to QPRAC+Proactive-EA, underscoring the effectiveness of our design.

Figure 15 shows the Alerts per tREFI for our QPRAC implementations. QPRAC-NoOp incurs almost one Alert every tREFI, leading to considerable performance degradation. It drops more than 20% performance in many memory-intensive applications, such as *429.mcf* and *482.sphinx3*, and a maximum of 46% performance drops in *510.parest* due to frequent Alerts, occurring more than two Alerts per tREFI. In contrast, QPRAC, which has opportunistic mitigations, significantly reduces the number of Alerts to 0.07 per tREFI. Finally, QPRAC+Proactive-EA, our default design, incurs no overhead because proactive mitigation during REF eliminates the Alerts.

### B. Sensitivity to Number of RFMs per Alert

PRAC specification allows a predefined number of All-Bank RFMs (1, 2, or 4) to be issued per Alert. Our default is 1 RFM per Alert, but using 2 or 4 RFMs per Alert can further reduce $T_{RH}$. Figure 16 shows the performance overheads of QPRAC under PRAC-1, PRAC-2, or PRAC-4 configurations (1, 2, or 4 RFMs per Alert) compared to the baseline. QPRAC alone incurs 0.8%, 0.8%, and 0.9% slowdown at 1, 2, and 4 RFMs per Alert respectively. In contrast, QPRAC with proactive mitigations–QPRAC+Proactive, QPRAC+Proactive-EA, and QPRAC-Ideal–incur no overhead, as proactive mitigations during REFs eliminate ABO occurrences.

The performance overhead remains similar across PRAC-1 to PRAC-4. This is because, although RFMs per Alert from 1 to 4 increases the mitigation cost per Alert (*i.e.*, the banks are blocked for a longer period), it also reduces the Alert frequency significantly. For example, compared to PRAC-1, PRAC-2 and PRAC-4 decrease Alerts by 1.9× and 3.3×, respectively. As a result, the slowdowns remain consistent.

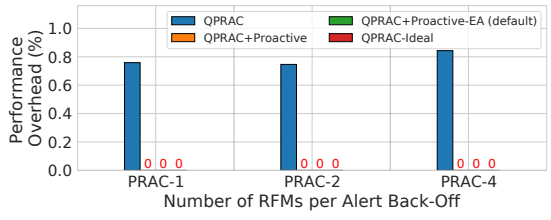



Fig. 16. Slowdown of QPRAC for RFMs/Alert values of 1, 2, and 4 (default = 1). QPRAC experiences a slowdown of 0.8% to 0.9%, while QPRAC with proactive mitigations–QPRAC+Proactive, QPRAC+Proactive-EA, and QPRAC-Ideal–incur no overhead for 1 to 4 RFMs per Alert.

### C. Sensitivity to Service Queue Size

Figure 17 shows the QPRAC performance as the priority-based service queue (PSQ) sizes vary from 1 to 5. QPRAC's performance remains consistently low, with less than 1% overhead across all evaluated queue sizes, showing slightly better performance at larger sizes. A 5-entry queue is used by default for QPRAC to ensure compatibility with the PRAC specification, supporting up to a $N_{mit}$ of 4 (i.e., PRAC-4).

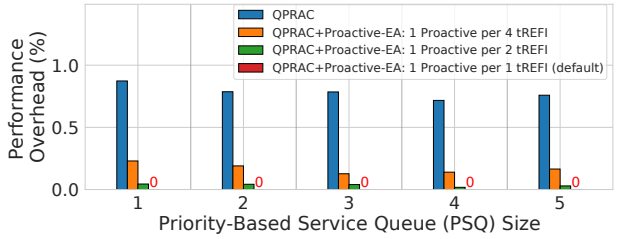



Fig. 17. Slowdown of QPRAC as the queue size varies for different proactive mitigation frequencies. QPRAC shows negligible overhead ($< 1\%$) across all evaluated queue sizes, with slightly better performance at larger sizes.

### D. Sensitivity to Back-Off Threshold

Figure 18 shows the performance of QPRAC as the Back-Off threshold ($N_{BO}$) varies from 16 to 128. Higher $N_{BO}$ values reduce the number of Alerts and limit slowdown but also increase the $T_{RH}$ tolerated by the defense. For $N_{BO}$ of 32 or more, QPRAC incurs negligible slowdown of less than 0.8%, while our designs with proactive mitigations, QPRAC+Proactive and QPRAC+Proactive-EA (our default), incur no slowdown, similar to QPRAC-Ideal. At $N_{BO}$ of 16, QPRAC has 2.3% slowdown, while QPRAC+Proactive, QPRAC+Proactive-EA, and QPRAC-Ideal, have less than 0.3% slowdowns. Although reducing $N_{BO}$ from 32 to 16 increases slowdowns for QPRAC from 0.8% to 2.3%, it only reduces $T_{RH}$ from 71 to 57. Therefore, we recommend a default $N_{BO}$ of 32 for QPRAC to ensure negligible slowdown even when proactive mitigation is unavailable.

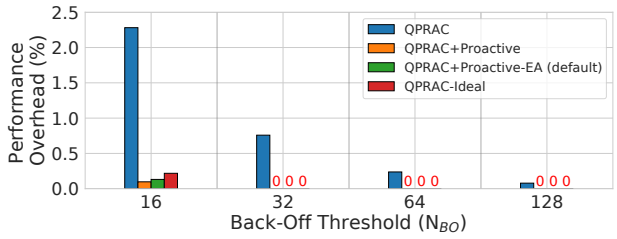



Fig. 18. Performance overhead of QPRAC as the Back-Off Threshold ($N_{BO}$) varies. QPRAC incurs a 0.9% slowdown at $N_{BO}$ of 32 and 2.9% at $N_{BO}$ of 16. In contrast, QPRAC designs with proactive mitigations show negligible slowdown across all evaluated $N_{BO}$ values, with no overhead at $N_{BO}$ of 32 and 0.2% at $N_{BO}$ of 16.

### E. Resilience to Performance Attacks

PRAC is vulnerable to performance attacks where an attacker induces a high rate of Alerts, reducing activation bandwidth for benign workloads. Figure 19 shows the worst-case bandwidth loss for QPRAC at different $N_{BO}$ in a multi-bank attack, where simultaneous hammering across N banks triggers a stream of Alerts and RFM-induced bandwidth loss.

As per the PRAC specification, All-Bank RFM ($RFM_{ab}$) is used on Alerts (which penalizes all 32 DRAM banks) since the interface does not identify which bank caused the Alert. Consequently, QPRAC-$RFM_{ab}$ suffers a high loss of activation bandwidth (62% to 93%) as $N_{BO}$ decreases from 128 to 16. With proactive mitigation, QPRAC-$RFM_{ab}$+Proactive can avoid bandwidth loss at $N_{BO}$ of 128, and limit it to 10% at $N_{BO}$ of 64, as it proactively mitigates rows before they reach activation counts of $N_{BO}$. However, at lower $N_{BO}$ of 32 and 16, this design suffers a bandwidth loss of 77% and 91%.

To prevent performance attacks at such $N_{BO}$, the PRAC specification can be modified to issue RFMs selectively rather than penalizing all banks with $RFM_{ab}$. $RFM_{sb}$ (same-bank), which mitigates one bank in each of the eight bank groups, reduces bandwidth loss to 42% and 68% at $N_{BO}$ of 32 and 16, respectively. $RFM_{pb}$ (per bank), a new command that mitigates a single bank, further reduces bandwidth loss to 15% and 27% at $N_{BO}$ of 32 and 16. These modifications require changes to the DRAM interface to communicate the mitigation needs to the host and require updates to the DRAM specification.

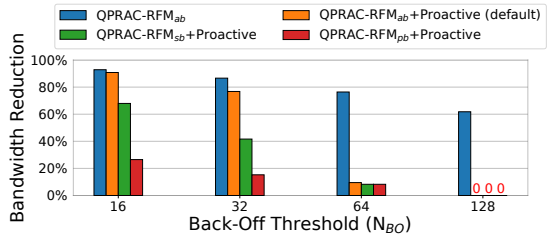



Fig. 19. Maximum DRAM Activation Bandwidth (BW) reduction under attack for QPRAC. $RFM_{ab}$ (all-bank RFM) suffers severe BW degradation, while proactive mitigation addresses this up to $N_{BO}$ of 64. We note that $RFM_{sb/pb}$ (same-bank/per-bank RFM) is needed at lower $N_{BO}$.

### F. Storage and Energy Overheads

We evaluate QPRAC's area overhead, static power, access energy, and circuit latency using the Synopsys Design Compiler with a 45nm process. The area overhead is scaled down to DRAM 10nm process to align with our baseline chip [5].

**Storage and Latency**: QPRAC uses a 5-entry priority-based service queue (PSQ) per DRAM bank by default for the mitigation rate of 1 RFM per Alert. Each entry has a 7-bit activation counter and a 17-bit RowID; together, this requires an SRAM storage of 15 bytes per DRAM bank. Additionally, each DRAM row is accompanied by an activation counter specified by PRAC – we use 7-bit counters per row. This incurs $0.038mm^2$ overhead, taking 0.05% of a single DDR5 chip [5]. PSQ operations, such as counter increment, comparison, and insertion, take 2.5ns with a 45nm CMOS process and occur in the shadow of Precharge (36ns), introducing no overhead.

**Energy**: Our synthesis results show the logic for QPRAC's PSQ operations (counter increment, comparison, and insertion) consumes an extra 0.23 $\mu$J per ACT, just 0.05% of the activation energy based on the Micron power calculator [42]. The QPRAC PSQ, smaller than the TRR trackers in the current DRAM [16], consumes a static power of 0.38 mW per chip.

Table III shows the energy overhead due to the issued mitigations for the different QPRAC designs, as the RFMs

per Alert (PRAC Level) is varied. QPRAC incurs a minimal energy overhead of 1.5% for all PRAC levels. While QPRAC+Proactive incurs no slowdown, its approach of proactively mitigating the highest activated row in the PSQ of each bank on *every* REF results in a significant energy overhead of 14.6%. In contrast, the energy-optimized design, QPRAC+Proactive-EA, reduces the energy overhead to 1.9% by performing proactive mitigations *only* when the highest activated row in the PSQ has an activation count that reaches or exceeds the proactive mitigation threshold ($N_{PRO}$), while ensuring negligible slowdown. Overall, QPRAC and QPRAC+Proactive-EA incur negligible energy overheads.

TABLE III
ENERGY OVERHEAD OF QPRAC

| PRAC Level | QPRAC | QPRAC+Proactive | QPRAC+Proactive-EA |
|---|---|---|---|
| PRAC-1 | 1.2% | 14.6% | 1.9% |
| PRAC-2 | 1.3% | 14.6% | 1.9% |
| PRAC-4 | 1.5% | 14.6% | 1.9% |

### G. Comparison with In-DRAM Mitigations

Figure 20 compares the performance of QPRAC against state-of-the-art in-DRAM Rowhammer mitigations, Mithril [26] and PrIDE [19], as the Rowhammer threshold ($T_{RH}$) varies. We assume at most one proactive mitigation per REF and use the DRAM timings for Mithril and PrIDE without PRAC-specific timing increases [40]. All schemes show minimal slowdown at $T_{RH}$ of 1024. However, at ultra-low $T_{RH}$ ($T_{RH} \leq 512$), both Mithril and PrIDE incur significant slowdowns. Mithril's performance drops by 69%, 54%, 32%, and 10%, at $T_{RH}$ of 64, 128, 256, and 512, while PrIDE shows 54%, 32%, 19%, and 7% slowdowns at the same $T_{RH}$. In contrast, QPRAC incurs no slowdown across all evaluated thresholds. Additionally, Mithril requires a 5,300-entry CAM/bank, which is impractical, whereas QPRAC only requires a 5-entry CAM/bank, smaller than the TRR trackers in DDR4 [16].

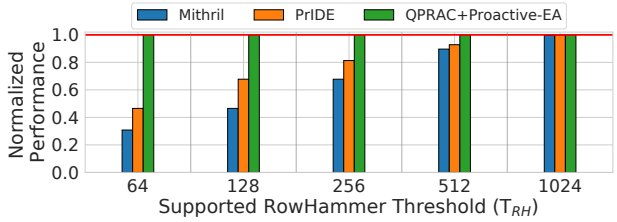



Fig. 20. Normalized performance of QPRAC, Mithril, and PrIDE as the Rowhammer threshold ($T_{RH}$) varies. Mithril and PrIDE incur significant overhead at ultra-low $T_{RH}$ ($T_{RH} \leq 512$). Mithril's performance drops from 69% to 10% as $T_{RH}$ increases from 64 to 512, while PrIDE shows slowdowns ranging from 54% to 7% at the same thresholds. In contrast, QPRAC incurs no performance overhead across all evaluated $T_{RH}$.

## VII. RELATED WORK

**A. Secure PRAC Designs:** A concurrent work, MOAT [46], also demonstrated vulnerabilities with PRAC. MOAT showed that in a PRAC implementation like Panopticon, configured for a Rowhammer threshold of 128, tardiness in mitigation using FIFO-based queues can cause activation counts of up to 1150, far beyond the Rowhammer threshold of 128. In comparison, we demonstrate much worse attacks exploiting FIFO-based queues, achieving up to 1300 activations by exploiting $ABO_{ACT}$ fundamental to PRAC specification (*Fill+Escape* attack) and up to 30,000 ACTs exploiting the t-bit toggling in Panopticon (*Toggle+Forget* attack).

Figure 21 compares the performance of MOAT and QPRAC for PRAC-1 (1 RFM per Alert) as $N_{BO}$ varies. We assume MOAT uses an enqueuing threshold of $N_{BO}/2$ and a single-entry queue with an additional register, as described in their work [46]. Both QPRAC and MOAT show negligible slowdown (<1%) at $N_{BO}$ of 32 or higher due to the infrequent Alerts. However, due to its multi-entry queue design, QPRAC outperforms MOAT at lower $N_{BO}$, benefiting from the opportunistic and proactive mitigations. For example, at $N_{BO}$ of 16, MOAT and its variant with at most one proactive mitigation per 4 tREFI and at most 1 per tREFI shows 3.6%, 2.5%, and 0.7% slowdown. In contrast, QPRAC and its variants with the same proactive mitigation ratios have only a slowdown of 2.3%, 1.2%, and 0.1%, demonstrating better scalability.

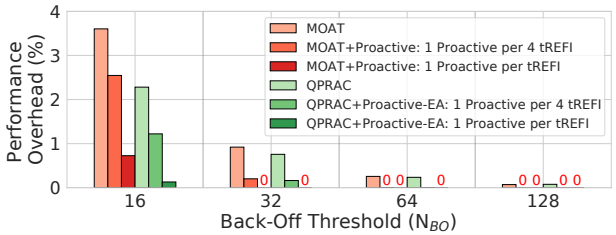



Fig. 21. Slowdown of MOAT [46] and QPRAC as the Back-Off threshold ($N_{BO}$) varies. At $N_{BO}$ of 32 or more, MOAT and QPRAC show negligible overhead (less than 1%). At $N_{BO}$ of 16, MOAT and MOAT with 1 proactive mitigation per tREFI incur slowdown of 3.6% and 0.7%, while QPRAC and its variant with proactive mitigation have lower slowdowns of 2.3% and 0.1%.

Figure 22 shows the energy overhead of MOAT and QPRAC as $N_{BO}$ varies. At $N_{BO}$ of 32 or higher, both MOAT and QPRAC exhibit negligible energy overhead of less than 2%, due to the dual-threshold design (MOAT) and energy-aware proactive mitigation (QPRAC). At $N_{BO}$ of 16, MOAT and MOAT with one proactive mitigation per tREFI incur a 5.7% and 5.1% energy overhead. In contrast, QPRAC and its variant with the same proactive mitigation ratio incur a 4.1% and 4.6% overhead, due to QPRAC's reduced execution time.

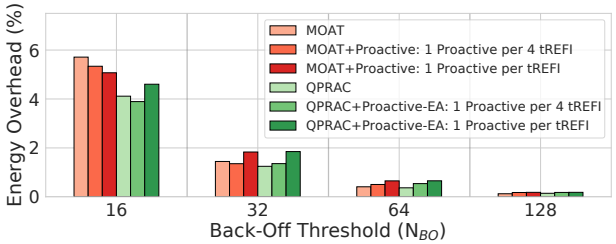



Fig. 22. Energy overhead of MOAT and QPRAC as the Back-Off threshold ($N_{BO}$) varies. For $N_{BO}$ of 32 or higher, both QPRAC and MOAT show negligible energy overhead (less than 2%) due to the dual-threshold design (MOAT) and energy-aware proactive mitigation (QPRAC). At $N_{BO}$ of 16, MOAT and MOAT with one proactive mitigation per tREFI incur 5.7% and 5.1% overhead, while QPRAC incurs 4.1% and 4.6% overhead, respectively.

**B. One Counter-Per-Row**: PRHT [28] maintains per-row counters in DRAM to identify aggressor rows, like PRAC [2], [4]; however, it reports a 10% failure rate, suggesting it is insecure. In comparison, QPRAC has deterministic security at $T_{RH}$ as low as 71. Hydra [48], CRA [23], and START [52] store per-row counters in a reserved DRAM region and require extra DRAM accesses by the memory controller to fetch them, causing high worst-case slowdowns at sub-100 $T_{RH}$. In contrast, we use PRAC-based in-DRAM counters, avoiding such overheads and memory-controller changes.

**C. Efficient Aggressor Counting**: Other works use low-cost in-DRAM trackers using SRAM in DRAM chips, such as CAT [56], TWiCE [33], and Mithril [26] and ProTRR [38] (which use Misra-Gries summaries [45]). We compare their storage with QPRAC in Table IV. These solutions, feasible at $T_{RH}$ of 4K, requiring KBs of storage, become impractical at $T_{RH}$ below 100 as they require MBs of SRAM.

Samsung's DSAC [17] and Hynix's PAT [28] propose probabilistic trackers with less than 20 entries, but DSAC is insecure [18], and PAT has a similar failure rate as conventional trackers. PrIDE [19], a probabilistic in-DRAM tracker with four entries per bank, offers security up to $T_{RH}$ of 400 but suffers up to 30% bandwidth loss at $T_{RH}$ of 250. In comparison, QPRAC requires negligible storage (15 bytes per bank), has deterministic security, and incurs no overhead at sub-100 $T_{RH}$, surpassing all prior works.

TABLE IV
PER-BANK SRAM OVERHEAD OF IN-DRAM TRACKERS

| Name | $T_{RH}$ = 4K | $T_{RH}$ = 100 |
|---|---|---|
| Misra-Gries [45] | 42.5 KB | 1700 KB |
| TWiCe [33] | 300KB | 12 MB |
| CAT [56] | 196 KB | 7.84 MB |
| **QPRAC** | **15 bytes** | **15 bytes** |

**D. Alternative Mitigative Actions**: Memory-controller-based defenses can issue mitigations probabilistically [23], [25], [29], [57], [67]. However, these require knowledge of DRAM neighbors, which may be scrambled within DRAM, or solutions like Directed RFM (DRFM), whose rates are insufficient for sub-100 thresholds. Other methods involve row migration [15], [50], [51], [54], [62], [64] or access rate limiting [65], which introduce high overheads at $T_{RH}$ below 100. Other approaches [39], [66] modify the DRAM interface to allow simultaneous refresh or mitigation during activations. In contrast to all these proposals, our solution, QPRAC is in-DRAM, avoids intrusive changes, and maintains low overheads at $T_{RH}$ below 100.

**E. Error Detection and Correction**: SafeGuard [9], CSI-RH [22], PT-Guard [53], and MUSE [37] use message authentication codes to detect Rowhammer failures. Failures may also be reduced by scrambling data layouts with ECC [27]; however, uncorrectable bit-flips can still cause data loss. In comparison, QPRAC provides deterministic security against Rowhammer attacks, preventing any possibility of data loss.

## VIII. CONCLUSION

To enable scalable and secure Rowhammer mitigation, the recent JEDEC DDR5 specification introduces Per Row Activation Counting (PRAC) but provides minimal implementation guidelines. Existing approaches are either insecure or impractical. Our paper addresses these challenges by proposing a PRAC implementation with a scalable priority-based service queue (PSQ) design, ensuring strong security at $T_{RH}$ below 100. Our solution, QPRAC, leverages opportunistic and proactive mitigations, achieving 0% slowdown at $T_{RH}$ of 71 while requiring just 15 bytes per bank.

## IX. ACKNOWLEDGMENTS

We thank Stefan Saroiu for sharing insights on PRAC in his keynote at DRAMSec'24 that inspired this work. We also thank Alec Wolman for insightful discussions on Panopticon and PRAC. Special thanks to Kuljit Bains for the feedback and valuable insights on PRAC. We are grateful to SK Hynix, Micron, and Samsung for their feedback, especially for inspiring the energy-optimized design of QPRAC. We also thank Kwangrae Kim for his assistance in the area and power analysis. This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) funding numbers RGPIN-2019-05059 and RGPIN-2023-04796.

## APPENDIX A
## PANOPTICON BLOCKING $ABO_{ACT}$ FROM TOGGLING T-BIT

As Panopticon suffers insecurity due to t-bit toggles by $ABO_{ACT}$ (*Toggle+Forget Attack* in Section II-E1), one way to address this can be to disallow $ABO_{ACT}$ from toggling t-bit.

However, now, an adversary can use $ABO_{ACT}$ to hammer a target row without mitigation. For a mitigation threshold of M and queue size of Q, an attacker could hammer M - 1 ACTs to Q rows in all 32 banks in parallel and then hammer Q rows $\times$ 1 ACT per bank, get an Alert and 3 $ABO_{ACT}$ to hammer the target row, repeating this for 32 banks. Figure 23 shows the results of this attack. At a minimum, this achieves 1800 ACTs unmitigated for M of 1024.

One could also disallow $ABO_{ACT}$ from toggling t-bit *only* for the bank with a full service queue. However, *Fill+Escape Attack* in Section II-E1 is still possible, where the attacker uses $ABO_{ACT}$ while the queue is full to induce Rowhammer, causing 1283 unmitigated activations to a target row. Thus, Panopticon is still insecure below $T_{RH}$ of 1200.

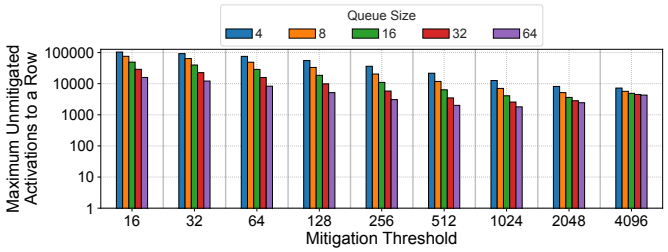



Fig. 23. The security vulnerability of Panopticon with blocking $ABO_{ACT}$ from toggling the threshold bit.

## APPENDIX B
## ARTIFACT APPENDIX

### A. Abstract

This artifact addresses two main aspects of the paper's results: (1) Security analysis of previous FIFO-based PRAC implementations, specifically targeting the Panopticon's t-bit toggle and our priority-based service queue implementation, QPRAC; (2) Performance analysis of QPRAC.

We provide Python scripts for the security analysis to evaluate the security metrics in Equation (2) and Equation (3). These scripts also regenerate the results presented in Figure 2 and Figures 6 to 13.

For the performance analysis, we offer (1) the C++ code for the QPRAC implementation, integrated with Ramulator2 [36], and (2) all evaluated workload traces and key evaluation results from Figure 14 to Figure 20. Additionally, Bash and Python scripts are provided to automate the collation and plotting of the results.

### B. Artifact Check-List (Meta-Information)

*1) Security Evaluations:*

- **Program:** Python3 programs to evaluate Equation (2) and Equation (3) for different QPRAC configurations. Bash scripts and Python scripts to generate evaluation results and produce plots.
- **Run-time environment:** Tested on Ubuntu 20.04 and 22.04 and should run on any Linux Distribution with a valid Python3 interpreter.
- **Hardware:** Single Core CPU desktop/laptop suffices.
- **Output:** Insecurity of Panopticon: Figure 2 and QPRAC security evaluation: Figure 6 to Figure 13
- **Experiments:** Instructions to run the experiments and parse the results are available in the provided README file.
- **How much disk space required (approximately)?:** Less than 100MB
- **How much time is needed to prepare workflow (approximately)?:** Under 5 minutes to install the dependencies.
- **How much time is needed to complete experiments (approximately)?:** $\approx$ 2 hours.
- **Publicly available?:** Yes, GitHub: https://github.com/sith-lab/qprac.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.14336354

*2) Performance Evaluations:*

- **Program:** C++ programs for Ramulator2, including QPRAC implementations, and Bash and Python scripts to run experiments, collate results, and generate plots.
- **Compilation:** Tested with g++ version 12.4.0; should also compile with any C++20-compliant compiler.
- **Run-time environment:** We suggest using a Linux distribution compatible with g++-10 or newer for the performance evaluations. For example, Ubuntu 22.04 or later is recommended if you prefer Ubuntu. This artifact has been tested on Ubuntu 22.04 and Rocky Linux 9.4.
- **Hardware:** We recommend using a CPU with at least 40 cores and 128GB or more of memory.
- **Metrics:** Weighted Speedup.
- **Output:** QPRAC performance results: Figures 14 to Figure 20.
- **Experiments:** Instructions for running experiments and parsing results are available in the provided README file.
- **How much disk space required (approximately)?:** $\approx$ 10GB.
- **How much time is needed to prepare workflow (approximately)?:** Under 10 minutes to install the dependencies and download the traces.
- **How much time is needed to complete experiments (approximately)?:**
  - $\approx$ 16 hours (on a cluster with 1000 cores) and $\approx$ 1 day (on an Intel Xeon CPU with 40 cores and 128GB memory) for the main results (Figures 14 and 15).
  - $\approx$ 2 days (on a cluster with 1000 cores) and $\approx$ 1 week (on an Intel Xeon CPU with 40 cores and 128GB memory) for all experiments (Figures 14 to 20).
- **Publicly available?:** Yes. Traces: https://zenodo.org/records/14607144. GitHub: https://github.com/sith-lab/qprac.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.14336354

### C. Description

*1) How to access:* The artifact is available at https://github.com/sith-lab/qprac.

*2) Hardware dependencies:*

- **Security Evaluation:** A single-core CPU desktop/laptop will allow to perform security analysis within $\approx$ 2 hours.
- **Performance Evaluation:** We strongly recommend using Slurm with a cluster capable of running bulk experiments to accelerate evaluations. If you opt for a personal server, we advise using a CPU with at least 40 cores and 128GB or more memory.

*3) Software dependencies:*

- **Security Evaluation:**
  - Python3 (Tested on V3.11.5).
  - Python3 Package matplotlib (v.3.4.0 or higher is required) for plotting.
- **Performance Evaluation:**
  - g++ with C++20 support (tested with version 12.4.0).
  - Python3 (recommended: version 3.10 or above).

*4) Traces:* We use 57 traces from SPEC2006, SPEC2017, TPC, Hadoop, MediaBench, and YCSB, available for download at https://zenodo.org/records/14607144.

### D. Installation and Experiment Workflow

First, clone the GitHub repository:

```
$ git clone https://github.com/sith-lab/qprac.git
```

*1) Security Evaluation:* No additional setup is required if dependencies are satisfied.

To start the experiment:

```
$ cd qprac/security_analysis
$ bash ./run_artifact
```

To use provided sample data and not regenerate results:

```
$ cd qprac/security_analysis
$ bash ./run_artifact –use-sample
```

*2) Performance Evaluation:* 1. Configure the following parameters in `perf_analysis/run_artifact.sh`:

- Using Slurm:
  - SLRUM_PART_NAME: Partition name for Slurm jobs.
  - SLRUM_PART_DEF_MEM: Default memory size for jobs (recommended: $\geq$ 4GB).
  - SLRUM_PART_BIG_MEM: Memory size for jobs that require large memory (recommended: $\geq$ 12GB).
  - MAX_SLRUM_JOBS: Maximum number of Slurm jobs to submit.
- Using a Personal Server:
  - PERSONAL_RUN_THREADS: Number of parallel threads to use for simulations.

2. Run the following commands to install dependencies, build Ramulator2, and execute simulations. If using a personal server with limited resources (e.g., less than 256GB memory or 40 cores), we recommend running only the main experiments first to avoid long execution times and then running the remaining experiments.

2.1. Running main experiments (Figures 14 and 15):

- Using Slurm:

```
$ cd qprac/perf_analysis
$ ./run_artifact.sh --method slurm --artifact main
```

- Using a Personal Server:

```
$ cd qprac/perf_analysis
$ ./run_artifact.sh --method personal --artifact main
```

2.2. Running all experiments (Figures 14 to 20):

- Using Slurm:

```
$ cd qprac/perf_analysis
$ ./run_artifact.sh --method slurm --artifact all
```

- Using a Personal Server:

```
$ cd qprac/perf_analysis
$ ./run_artifact.sh --method personal --artifact all
```

### E. Evaluation and Expected Results

*1) Security Evaluation:* The artifact provides the following scripts in the qprac/security_analysis/analysis_scripts directory: `equation2.py`, `equation2_pro.py`, `equation3.py`, `equation3_pro.py`, and `tbit_attack.py`. These scripts allow for the collation of results, with commands for generating the results for Figure 2, Equation (2), and Equation (3) provided in the `run_artifact.sh` script and the README file. After running `run_artifact.sh`, the t-bit attack results, along with the results for Equation (2) and Equation (3) in both QPRAC and QPRAC + Proactive, will be available as `tbit_attack.txt`, `PRAC1-4.txt`, `PRAC1-4_PRO.txt`, `R1.txt`, and `R1_PRO.txt` files in the qprac/security_analysis directory. Additionally, the regenerated figures, including Figure 2 and Figure 6 to Figure 13, can be found as `figure#.pdf` files in the corresponding qprac/security_analysis/figure# folders. Sample data required to generate each figure is provided in the qprac/security_analysis/figure#/sample_data folders.

*2) Performance Evaluation:* After completing the experiments using `run_artifact.sh`, the results and plots can be regenerated with the provided scripts. Specifically, the artifact includes the `plot_main_figures.sh` and `plot_all_figures.sh` files in the qprac/perf_analysis directory. These scripts collate the results (obtained as CSV files in qprac/perf_analysis/results/csvs) and generate the plots (obtained as PDF files in qprac/perf_analysis/results/plots). The `plot_main_figures.sh` script regenerates Figures 14 and 15, while the `plot_all_figures.sh` script generates Figures 14 to 20. Additionally, we provide scripts to collate results (`generate_csv_fig#.py`) and generate plots (`plot_fig#.py` or `plot.ipynb`) for each experiment in the qprac/perf_analysis/plot_scripts directory. Sample result files and plots are available in the qprac/perf_analysis/results/sample_results directory.

### F. Experiment Customization: Performance Evaluation

We offer easy configuration options for the following parameters: 1) the evaluated QPRAC mechanisms, 2) the evaluated Back-Off Thresholds ($N_{BO}$), 3) the tested PRAC levels (number of RFMs per alert), and 4) the simulation duration (minimum number of instructions per core during experiments).

These parameters can be customized in the `qprac/perf_analysis/sim_scripts/run_config_fig#.py` files:

- Mitigation list, PSQ sizes, and proactive mitigation frequencies: `mitigation_list`, `psq_sizes`, and `targeted_ref_ratios`.
- Back-Off Thresholds: `NBO_lists`.
- PRAC levels: `PRAC_levels`.
- Simulation duration: `NUM_EXPECTED_INSTS`.

### G. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae

## References


[1] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," in *ASPLOS*, 2016.

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

[3] 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.

[4] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, O. Ergin, and O. Mutlu, "Understanding the security benefits and overheads of emerging industry solutions to dram read disturbance," in *Workshop on DRAM Security (DRAMSec)*, 2024.

[5] I. Choi, S. Hong, K. Kim, J.-S. Hwang, S. Woo, Y.-S. Kim, C.-R. Cho, E.-Y. Lee, H.-J. Lee, M.-S. Jung, H.-Y. Jung, J.-S. Hwang, J. Yoon, W. Lim, H.-J. Yoo, W.-K. Lee, J.-K. Oh, D.-S. Lee, J.-E. Lee, J.-H. Kim, Y.-K. Kim, S.-J. Park, B.-K. Ho, B.-W. Na, H.-I. Choi, C.-K. Lee, S.-J. Lee, H. Shin, Y.-K. Lee, J.-W. Ryu, S. Shin, S. Park, D. Lim, S.-J. Bae, Y.-S. Sohn, T.-Y. Oh, and S. Hwang, "13.2 a 32gb 8.0gb/s/pin ddr5 sdram with a symmetric-mosaic architecture in a 5th-generation 10nm dram process," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 234–236.

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

[7] 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*, 2010, pp. 143–154.

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

[9] 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 *HPCA*. IEEE, 2022.

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

[11] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the many sides of target row refresh," in *IEEE Symposium on Security and Privacy*, 2020.

[12] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, "Mediabench ii video: Expediting the next generation of video systems research," *Microprocessors and Microsystems*, vol. 33, no. 4, pp. 301–318, 2009.

[13] 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 *IEEE Symposium on Security and Privacy*, 2018.

[14] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.

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

[16] 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*, 2021, pp. 1198–1213.

[17] 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.

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

[19] A. Jaleel, G. Saileshwar, S. Keckler, and M. Qureshi, "Pride: Achieving secure rowhammer mitigation with low-cost in-dram trackers," in *Annual International Symposium on Computer Architecture*, 2024.

[20] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," in *43rd IEEE Symposium on Security and Privacy'22 (Oakland)*, 2022.

[21] 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 2024)*, 2024.

[22] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "Csi: Rowhammer-cryptographic security and integrity against rowhammer," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 236–252.

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

[24] 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 *ISCA*, 2020.

[25] 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.

[26] 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 *HPCA*, 2022.

[27] 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.

[28] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang, S. Lee, D. Kim, S. Ku, D. Choi, N. Joo, S. Yoon, J. Noh, B. Go, C. Kim, S. Hwang, M. Hwang, S.-M. Yi, H. Kim, S. Heo, Y. Jang, K. Jang, S. Chu, Y. Oh, K. Kim, J. Kim, S. Kim, J. Hwang, S. Park, J. Lee, I. Jeong, J. Cho, and J. Kim, "A 1.1v 16gb ddr5 dram with probabilistic-aggressor tracking, refresh-management functionality, per-row hammer tracking, a multi-step precharge, and core-bias modulation for security and reliability enhancement," in *ISSCC*, 2023.

[29] 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," *ISCA*, 2014.

[30] 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, 2015.

[31] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering from the next row over," in *USENIX Security Symposium*, 2022.

[32] 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)*. IEEE, 2020, pp. 695–711.

[33] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: preventing row-hammering by exploiting time window counters," in *ISCA*, 2019.

[34] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "Moesi-prime: preventing coherence-induced hammering in commodity workloads," in *ISCA*, 2022.

[35] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Rowpress: Amplifying read disturbance in modern dram chips," in *ISCA-50*, 2023.

[36] 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 Computer Architecture Letters*, vol. 23, no. 1, pp. 112–116, 2024.

[37] E. Manzhosov, A. Hastings, M. Pancholi, R. Piersma, M. T. I. Ziad, and S. Sethumadhavan, "Revisiting residue codes for modern memories," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 73–90.

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

[39] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023.

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

[41] "DDR5 SDRAM Datasheet: Directed Refresh Management (DRFM), Page-290," Micron Technology Inc., 2022. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/ddr5_sdram_core.pdf

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

[43] 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.

[44] 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
[45] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet lightweight row hammer protection," in *MICRO*. IEEE, 2020, pp. 1–13.
[46] 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 (ASPLOS)*, 2025.
[47] 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.
[48] 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 *ISCA*, 2022.
[49] SAFARI Research Group, "ABACuS — GitHub Repository," 2023. [Online]. Available: https://github.com/CMU-SAFARI/ABACuS
[50] 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 *ASPLOS*, 2022.
[51] 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*, 2024, pp. 1014–1028.
[52] A. Saxena and M. Qureshi, "Start: Scalable tracking for any rowhammer threshold," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 578–592.
[53] 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 *DSN*, 2023.
[54] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *MICRO*, 2022.
[55] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.
[56] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating wordline crosstalk using adaptive trees of counters," in *ISCA*, 2018.
[57] M. Son, H. Park, J. Ahn, and S. Yoo, "Making dram stronger against row hammering," in *Design Automation Conference*, 2017.
[58] "SPEC CPU2017 Benchmark Suite," Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2017/
[59] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "Go go gadget hammer: Flipping nested pointers for arbitrary data leakage," in *USENIX Security*, 2024.
[60] Transaction Processing Performance Council, "TPC Benchmarks." [Online]. Available: http://tpc.org/
[61] 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 *ACM-CCS*, 2016.
[62] M. Wi, J. Park, S. Ko, M. J. Kim, N. S. Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *HPCA*, 2023.
[63] 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.
[64] 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.
[65] A. G. Yağlikçi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA*, 2021.
[66] 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 *MICRO*, 2022.
[67] J. M. You and J.-S. Yang, "Mrloc: Mitigating row-hammering based on memory locality," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.