GPUThor: Amplifying Rowhammer Attacks via Non-Uniform Patterns to Exploit ECC-Protected GPUs

Chris S. Lin
University of Toronto
Toronto, Canada
shaopenglin@cs.toronto.edu

Joyce Qu
University of Toronto
Toronto, Canada
joyce.qu@mail.utoronto.ca

Aditya Rajeev
University of Toronto
Toronto, Canada
aditya.rajeev@mail.utoronto.ca

Gururaj Saileshwar
University of Toronto
Toronto, Canada
gururaj@cs.toronto.edu

Abstract

GDDR memory in GPUs is vulnerable to Rowhammer attacks, where rapid memory accesses induce bit flips in adjacent cells, enabling data tampering and privilege escalation. However, prior GPU Rowhammer attacks trigger only tens to hundreds of bit flips, orders of magnitude fewer than CPU attacks, severely limiting their practical impact. This gap stems from the reliance of existing GPU Rowhammer attacks on uniform hammering patterns that activate aggressor and dummy rows equally, which results in low hammering intensity for aggressor rows.

We present GPUThor, a high-intensity Rowhammer attack on NVIDIA GPUs leveraging non-uniform hammering. GPUThor reverse engineers GPU memory-access coalescing behavior to enable non-uniform hammering patterns on GPUs, that activate aggressor rows more intensely than dummy rows. Additionally, by identifying refresh instances when in-DRAM mitigations are applied, it constructs longer attack patterns that escape mitigation across refresh intervals, further increasing hammering intensity. Together, these techniques yield 500× to 23,500× more bit flips than prior GPU Rowhammer attacks, across several NVIDIA GPUs (A4000, A4500, A5000, A6000), reaching bit flip rates close to state-of-the-art CPU Rowhammer attacks. GPUThor also enables the first Rowhammer exploits on ECC-protected GPUs, inducing uncorrectable double and triple bit flips, making denial-of-service and privilege-escalation attacks practical even on GPUs with ECC enabled.

CCS Concepts

• **Security and privacy → Hardware attacks and countermeasures**.

Keywords

DRAM Rowhammer Attacks, GPU Security, GDDR DRAM

ACM Reference Format:
Chris S. Lin, Joyce Qu, Aditya Rajeev, and Gururaj Saileshwar. 2026. GPUThor: Amplifying Rowhammer Attacks via Non-Uniform Patterns to Exploit ECC-Protected GPUs. In *Proceedings of Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26).* ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

CCS '26, The Hague, The Netherlands

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Rowhammer is a read-disturbance vulnerability in DRAM that enables attackers to induce bit flips in memory cells by rapidly activating neighboring rows [47]. Such attacks have been shown to enable data tampering, sandbox escapes, and privilege escalation exploits [8, 11, 21, 35, 36, 91]. While these were first discovered over a decade ago in CPU-based DDR memories, in the past year, several new Rowhammer attacks have been demonstrated on GPU-based GDDR memories [31, 53, 55, 102]. GPUHammer [53] first discovered Rowhammer bit flips on NVIDIA A6000 GPUs with GDDR6 memory, and used them to degrade ML model accuracy. Subsequent works [31, 55, 102] further showed that Rowhammer on GPUs can even lead to system-wide privilege escalation, establishing GPU Rowhammer attacks as a potent threat to system security.

Despite these recent advances, the number of bit flips observed by recent GPU Rowhammer attacks remains *orders of magnitude lower* than what CPU-based attacks routinely achieve [29, 35], limiting their practicality. As shown in Table 1, GPUHammer [53] reported just 2 bit flips per DRAM bank on an A6000 GPU (16 flips per GB), while GeForge [102] and GPUBreach [55] reported 2.2 and 5.6 bit flips per bank (18 and 45 flips per GB) respectively on the same GPU. GDDRHammer [31] improved this to 94.8 bit flips per bank (758 flips per GB) using double-sided hammering patterns. In contrast, CPU-based Rowhammer attacks like Blacksmith [35] have demonstrated up to 550,000 flips per GB on DDR4 memories, orders of magnitude higher than the best GPU Rowhammer attacks. Due to the low bit flip rates on GPUs, enabling ECC on GDDR-based GPUs, which provides SECDED-level protections and corrects single-bit errors, effectively mitigates prior Rowhammer attacks. Accordingly, NVIDIA recommends enabling ECC as the primary defense against GPU Rowhammer attacks [70]. In this paper, we bridge the gap between GPU and CPU attacks, and demonstrate GPU Rowhammer attacks triggering up to *23,500× more bit flips* than prior attacks, and show that *ECC is an insecure mitigation for GPUs.*

Limitations of Prior Work. The fundamental limitation of prior GPU Rowhammer attacks is that they perform *uniform* hammering: all rows in the attack pattern, both aggressor rows (adjacent to the target victim) and decoy rows (used to evade in-DRAM mitigations), are activated at equal rates. Thus, a significant fraction of each refresh interval is spent hammering decoy rows rather than aggressors. In contrast, CPU Rowhammer attacks like Blacksmith [35] employ *non-uniform* hammering, activating aggressors at much higher intensity relative to decoys while still evading mitigations, dramatically increasing bit flips. No prior GPU Rowhammer attack has successfully utilized non-uniform hammering, leaving

the threat of Rowhammer on GPUs severely underestimated.[1] However, adapting non-uniform hammering to GPUs is non-trivial and requires addressing a few key challenges (**C1-C3**).

C1. Memory Access Coalescing. Unlike CPUs, we discover that GPUs have optimizations that aggressively coalesce memory accesses at multiple levels within warps and at the memory controller to amortize memory bandwidth. This coalescing prevents the fine-grained control over activations that non-uniform hammering requires: repeated accesses intended to hammer aggressors at higher intensity may be merged into a single DRAM activation, collapsing a naive non-uniform pattern back to a uniform one.

C2. Unknown Mitigation Instances. Non-uniform hammering can be more effective with the knowledge of *when* in-DRAM mitigations sample aggressors and issue mitigative refreshes, so that true aggressors can be hammered intensely outside the sampling window. In CPU DRAMs, these intervals are known to be aligned to multiples of tREFI [11, 35]. However, the sampling and mitigation instances in GDDR6 memories have not been characterized, leaving attackers unable to design effective non-uniform patterns.

C3. Defeating ECC. Even if hammering intensity is amplified, workstation-class GPUs (e.g., A6000) support SECDED ECC that can correct single-bit errors and detect double-bit errors in GDDR6 memory. This makes it difficult for Rowhammer attacks to be successful in the presence of ECC.

Our Approach. We introduce *GPUThor*, the first high-intensity, non-uniform Rowhammer attack on NVIDIA GPUs. Our approach is enabled by reverse-engineering the micro-architectural behavior of GPU memory accesses and GDDR6 in-DRAM mitigations.

First, to address memory request coalescing (C1), we systematically characterize the behavior of repeated memory requests on GPUs, within and across warps. We find that while repeated requests within a single warp are aggressively coalesced at the memory controller, requests across warps are typically not coalesced. Moreover, when targeting different cache lines from across warps, requests further have minimal interference at the cache level. Leveraging this, we design hammering kernels that distribute repeated accesses to a row across warps on *independent cache lines* within a row, ensuring distinct repeated activations. This yields a 2–3× increase in hammering intensity over prior works [31, 53, 55, 102].

Second, to overcome the unknown mitigation instances (C2), we reverse-engineer the Target Row Refresh (TRR) behavior on Ampere GPUs with GDDR6 memories. We discover that mitigations are issued approximately once every 72 tREFIs, rather than once per tREFI as assumed by prior works [53, 55]. Leveraging this, we develop non-uniform attack patterns spanning up to 6 tREFIs, with a majority of the time spent hammering an aggressor repeatedly, except for the final tREFI that solely uses decoy rows to overwhelm TRR. This increases hammering intensity, i.e., activation rates per aggressor, to nearly 6.6× that of prior uniform patterns [53].

Hammering Campaigns. We evaluate GPUThor on four Ampere-class GPUs (A4000, A4500, A5000, A6000), a broader set than prior works [31, 53, 55, 102]. Across four banks per GPU with ECC disabled, we observe **72,000–377,000 bit flips per GB**, up to 500× higher than GDDRHammer [31] and 23,500× higher than GPUHammer [53]. These flip rates approach the DDR4 bit flip rates with CPU attacks like Blacksmith (550,000 flips/GB) [35], indicating comparable vulnerability. At these bit-flip rates, even SECDED ECC, assumed to be applied at 16-byte granularity in workstation GPUs [94] is insufficient. Across these GPUs (four banks each), at 16-byte granularity, we discover **387 double-bit flips** that are detectable but uncorrectable errors (DUE) by the SECDED ECC, and **2 triple-bit flips** that the ECC can neither detect nor correct, resulting in silent data corruption (SDC); the most vulnerable GPU, the A5000, accounts for 306/387 double-bit flips and 2/2 triple-bit flips. We thus surpass the challenge of defeating ECC (C3) that limits prior GPU Rowhammer attacks.

Table 1: Number of bit flips across GPU Rowhammer attacks. GPUThor induces up to 500× and 23,500× more flips per GB than prior works, GDDRHammer and GPUHammer.*

Attack	GPU	Flips/Bank	Flips/GB	Ratio
GPUHammer [53]	RTX A6000	2.0	16	1×
GeForge [102]	RTX A6000	2.2	18	1.1×
GPUBreach [55]	RTX A6000	5.6	45	2.8×
GDDRHammer [31]	RTX A6000	94.8	758	47×
GPUThor (our work)	RTX A6000	**14,311**	**114,488**	**7,155×**
	RTX A5000	**23,597**	**377,552**	**23,597×**
	RTX A4500	**4,689**	**75,024**	**4,689×**
	RTX A4000	**4,548**	**72,768**	**4,548×**

*Bit flip counts for prior works are from their respective papers.

Implications for ECC. We launch campaigns with ECC enabled on the A6000 GPU (our A4000-5000 GPUs are cloud-based, where the provider does not give the option to enable ECC). On the A6000 with ECC enabled, we trigger detectable uncorrectable errors (DUE), i.e., double-bit errors, causing ECC failures and GPU crashes, at an average rate of 1 DUE every 2 hours; on the more vulnerable A5000 GPU, we estimate it would suffer one DUE per 20 minutes on average. Each DUE renders the GPU unusable and requires a GPU reset or a system reboot, enabling denial-of-service attacks, as recovery on cloud systems can take up to 20 minutes per DUE [9].

Crucially, we discover that enabling ECC does not prevent privilege escalation exploits; we observe exploitable multi-bit errors with ECC enabled. Triple-bit errors occurring while ECC was enabled cause the SECDED code to mis-correct, leading to silent data corruption (SDC) that allows the corrupted data to be consumed *without triggering a DUE that kills the GPU*. Moreover, we discover that even double-bit DUEs are exploitable, since DUEs are serviced lazily in NVIDIA GPUs, leaving a ~10 ms time window between DUE detection and the GPU being killed, during which the corrupted data is consumed by the attacker's GPU kernel. Exploiting these properties of SDC and DUEs, we demonstrate that *privilege escalation attacks* are indeed feasible on an A6000 GPU *even with ECC enabled*. These results show that NVIDIA-recommended defenses such as ECC are easily bypassed and stronger defenses are needed to mitigate GPU Rowhammer attacks.

[1] While GeForge [102] *claims* to perform non-uniform hammering, we observe that its access patterns collapse to close to uniform hammering, due to memory request coalescing (**C1**); it has similar bit-flip rates (2.2 per bank) to GPUHammer (2 per bank) [53] and lower than GPUBreach (5.6 per bank) [55] which uses uniform hammering.

Contributions. This paper makes the following contributions:

(1) We demonstrate GPUThor, the first non-uniform, high-intensity Rowhammer attack on NVIDIA GPUs, inducing 500× to 23,500× more bit flips than prior works.
(2) We reverse-engineer the coalescing behavior of GPU memory accesses and TRR mitigation instances to enable effective non-uniform, multi-tREFI hammering patterns on GPUs.
(3) Using GPUThor on GPUs like A4000, A4500, A5000, and A6000, we discover 72,000 to 377,000 bit flips/GB in GDDR6 DRAM with ECC disabled, and hundreds of multi-bit flips (387 double and 2 triple bit flips) uncorrectable by ECC.
(4) We demonstrate the first Rowhammer-based denial-of-service and privilege escalation attacks on ECC-protected GPUs.

Responsible Disclosure. We disclosed our findings to NVIDIA on 29 April 2026, and also to the major cloud vendors (Google, Microsoft, AWS). NVIDIA requested an embargo until 25th August 2026, when they plan to release a security bulletin providing consumer guidance. They did not share any details on mitigation plans.

2 Background

In this section, we discuss the threat model, the memory system, and the GDDR6 DRAM on NVIDIA GPUs, GPU Rowhammer attacks, and error correction codes (ECC) as a defense on NVIDIA GPUs.

2.1 Threat Model

We assume an unprivileged attacker who can launch CUDA kernels on an NVIDIA GPU with GDDR6 memory, such as A4000–A6000, popularly used in workstations and in the cloud. The attacker can co-locate with another victim user on a time-shared GPU in the cloud [22], and attempt to tamper with its data using Rowhammer, or run from an unprivileged process in a single-tenant setting and attempt privilege escalation to root by tampering GPU Page Tables via Rowhammer [31, 55, 102]. Like prior GPU Rowhammer attacks [31, 53, 55, 102], the attacker can reverse-engineer the GPU virtual-address-to-DRAM bank and row mappings using timing side-channels and hammer neighbors of target victim rows. The privilege escalation exploits in §8 assume IOMMU is enabled, matching prior work [55].

2.2 GPU Memory System

GPU Memory Hierarchy. As shown in Figure 1, discrete NVIDIA GPU chips have an array of Streaming Multiprocessors (SMs) that issue memory requests through an on-chip L1/L2 cache hierarchy to a set of on-die memory controllers (MC). Threads within a warp (a group of 32 threads) issue memory instructions in lockstep, and each MC in workstation GPUs (A4000–6000) is connected to independent GDDR6 DRAM channels [37]. The GPU memory system can *coalesce* multiple requests to the same L1 cache line within a warp to amortize its cost [71], although coalescing behaviors at other levels (L2 cache, and memory controller) are undocumented. As we show later (§4), such coalescing has implications for mounting Rowhammer attacks.

GDDR6 Organization. As shown in Figure 1, a GDDR6 device is organized as a hierarchy of channels, chips, banks, and rows [37]. Each channel is split into two independent subchannels (Channel A and Channel B), and each chip contains 16 banks; each bank contains thousands of rows of cells storing charge (1s and 0s), and all rows in a bank share a common *row buffer*. Data is accessed at the granularity of a row: an ACT command opens the addressed row and latches it into the row buffer, column reads/writes operate on the buffered row, and a PRE command precharges the bitlines before another row in the same bank can be activated.

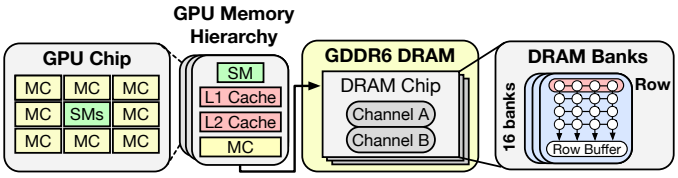

Figure 1: Memory system in NVIDIA GPUs with GDDR6 DRAM. SMs issue memory requests that are serviced by the L1 or L2 cache or the memory controllers (MC). Each memory controller is connected to a GDDR6 DRAM with two channels, and each DRAM chip has 16 banks; within a bank, DRAM cells are organized as rows, all sharing a common row buffer.

Refresh. DRAM cells leak charge and must be periodically refreshed to preserve their contents. Thus, the memory controller issues a REF command once every refresh interval tREFI ($\leq$1.9 μs in GDDR6), which refreshes a subset of rows, so that every row is refreshed within the refresh window tREFW (32 ms in GDDR6) [37]. REF commands also provide the DRAM the time to issue Rowhammer mitigation refreshes [21, 28]; so tREFI is also the typical time unit for constructing Rowhammer access patterns [11, 35, 53].

2.3 CPU Rowhammer Attacks and Defenses

Rowhammer. Rowhammer is a read-disturbance vulnerability in DRAM in which repeatedly activating ("hammering") a row causes charge leakage in electrically adjacent rows, ultimately flipping bits in those victim cells [47]. The minimum number of activations to a row to induce a bit flip is called the *Rowhammer threshold* (HC_{first}). Rowhammer access patterns can target a *victim* row by hammering one (*single-sided*) or both (*double-sided*) of its immediate neighbors, also called *aggressors*. On CPUs, Rowhammer has been used for a variety of exploits, including privilege escalation, sandbox escape, and cryptographic key recovery [8, 11, 23, 35, 36, 91, 99].

Target Row Refresh (TRR). Modern DRAM chips, starting from DDR4, deploy in-DRAM mitigations against Rowhammer, collectively referred to as Target Row Refresh (TRR). TRR samples frequently activated rows and issues a *mitigative refresh* to their neighbors to reverse the charge leakage in victim rows [21, 28]. However, as TRR trackers have finite capacity, subsequent attacks [21, 35] evaded TRR by hammering *aggressor* rows interleaved with a larger set of *decoy* rows, overwhelming the tracker so the aggressors escape mitigation. While TRRespass [21] used *uniform* patterns that hammered aggressors and decoys at the same rate, Blacksmith [35] developed *non-uniform* patterns that hammer aggressors at a higher rate than decoys, increasing hammering intensity and flipping bits on a wider range of devices, while also significantly increasing the number of bit flips (up to 550K flips per GB in DDR4 devices [35]). As TRR mitigation instances align with multiples of tREFI [28],

such attacks synchronize the hammering patterns with REF commands [11]. Recent attacks [36, 65] also demonstrate that such non-uniform attack patterns induce bit flips in DDR5 devices with on-die ECC.

2.4 GPU Rowhammer Attacks

GPUHammer [53] first showed that GPU Rowhammer attacks are practical on an NVIDIA A6000 GPU with GDDR6 memory. By reverse-engineering the mappings of virtual addresses to GDDR6 banks and rows, it developed multi-thread and multi-warp hammering kernels for hammering GDDR6 memories. As shown in Figure 2 (a), by using n-sided patterns, i.e., one aggressor and $n-1$ decoys per tREFI, and *uniform* hammering (1× aggressor activation per tREFI, same as decoys) like TRRespass [21], it defeated TRR to induce bit flips in GDDR6 memories. GPUHammer used these bit flips to tamper with ML-model weights and degrade the model accuracy. **GPUBreach** [55] used similar uniform hammering patterns to induce bit flips, and tampered with GPU page table entries to enable system-wide privilege escalation.

GDDRHammer [31] and **GeForge** [102] also demonstrated privilege escalation attacks, with IOMMU disabled. GDDRHammer extends GPUHammer's uniform hammering from single-sided to double-sided, as shown in Figure 2 (a), providing 34× more bit flips [31]. While it evaluated non-uniform patterns accessing aggressors with higher intensity and multi-tREFI patterns, it found the most bit flips with its *uniform* pattern (single tREFI, 2× aggressor activations per tREFI). GeForge *claims* to use a *non-uniform* hammering pattern that spans three tREFI periods, accessing an aggressor row up to 13 times across 3 tREFIs. However, as it repeats accesses to the same row within a warp, based on our analysis (§4), these accesses get *coalesced* to a few activations per tREFI, resulting in approximately uniform hammering like GPUHammer. The number of bit flips with GeForge is just 1.1× that of GPUHammer [53] and less than GPUBreach [55], which use uniform hammering.

As summarized in Table 1, all four prior GPU Rowhammer attacks use largely *uniform* hammering patterns and only have tens of bit flips per bank, much lower than prior CPU attacks [29, 35]. Given the low bit-flip rates in GPUs so far, enabling ECC on GPUs has been sufficient to protect against these attacks [31, 53, 55, 102].

2.5 Error Correction Codes on GPUs

ECC. Workstation-class NVIDIA GPUs (e.g., A4000–A6000) with GDDR6 memories support error correction codes (ECC) that provide SECDED (single-error-correct, double-error-detect) level protection [94]. When ECC is enabled, prior analysis [53, 94] suggests that these GPUs store a 2 B ECC for every 32 B data (a 6.25% memory overhead). Thus, at the granularity of 16 B data, a 1 B SECDED code can *correct* any single bit flip (single-bit error, SBE) and *detect* but not correct any two-bit flip (double-bit error, DBE). A DBE results in a detectable, uncorrectable error (DUE), while with ≥ 3 flips, the data can get mis-corrected, causing silent data corruption (SDC). SECDED ECC in CPU DRAM has been shown to be insecure against Rowhammer attacks, with exploits shown on ECC-DIMMs in DDR3 [8] and DDR4 [42], and on DDR5 with on-die ECC [36, 65]. However, thus far, on GDDR6, enabling ECC mitigates all existing GPU Rowhammer attacks. NVIDIA also recommends enabling ECC on GPUs to defend against GPU Rowhammer attacks [70].

Error Management. In the event of a correctable SBE, the NVIDIA GPU silently repairs the SBE and the kernel continues executing; in the background, a counter tracking corrected errors is incremented, which is readable by an unprivileged user process via `nvidia-smi`. When a DUE is detected on Ampere GPUs, all the running GPU kernels are aborted and the GPU is rendered unusable until it is reset [69]. Additionally, a DUE triggers *row remapping*, in which the affected row is permanently retired and replaced with a spare row on the next GPU reset; if the pool of spare rows is exhausted, the device raises a *row-remapping failure* flag, indicating the device qualifies for RMA (return-merchandise-authorization), marking it as defective [69]. This occurs in the event of 9 DUEs in a bank or after 512 DUEs across the entire GPU memory [69].

3 Overview of GPUThor

The goal of GPUThor is to engineer Rowhammer attacks on NVIDIA GPUs with high hammering intensity, capable of producing orders of magnitude more bit flips than prior GPU attacks. We seek to evaluate the real vulnerability of these GPUs and whether ECC-based defenses recommended by NVIDIA [70] are truly secure. We achieve this by enabling *non-uniform* hammering on GPUs inspired by state-of-the-art CPU Rowhammer attacks [35, 36, 65].

Approach. To mount high-intensity GPU Rowhammer attacks via non-uniform patterns, GPUThor relies on two key building blocks:

(1) Defeating GPU memory request coalescing (§4). Since the GPU memory subsystem aggressively coalesces accesses within warps, achieving repeated DRAM activations for the same row is challenging for GPU Rowhammer attacks, unlike on CPUs. We characterize coalescing of memory requests at three levels (within a warp, across warps to the same address, and across warps to different cachelines of the same row) and find that the third regime reliably preserves repeated activations. Therefore, our hammering kernels distribute repeated accesses to an aggressor row across multiple warps *and* across unique cachelines of that row, enabling repeated DRAM activations and non-uniform hammering.

(2) Multi-tREFI non-uniform hammering (§5). While single tREFI hammering enables non-uniform patterns, there is a limit on how much the intensity of aggressors can be increased with a single tREFI pattern, as the placement of decoy rows with respect to TRR sampling instances becomes constrained. To address this, we develop multi-tREFI non-uniform hammering patterns. We characterize the mitigation instances on GDDR6 chips in Ampere GPUs, constructing non-uniform patterns spanning multiple tREFIs and using the resulting bit flip reproducibility as a side channel to learn TRR mitigation frequency. We find that TRR mitigation is performed approximately once every 72 tREFIs, rather than once per tREFI [53, 55]. We enable non-uniform hammering patterns with length up to 6 tREFIs and 6.6 aggressor activations per tREFI.

Combining (1) and (2) gives the GPUThor pattern shown in Figure 2 (b): a hammering pattern spanning 6 tREFIs, with the first 5 tREFIs having a total of 40 aggressor activations, and the final tREFI reserved for decoys to overwhelm TRR. Next, we describe these building blocks for our hammering pattern (§4, §5).

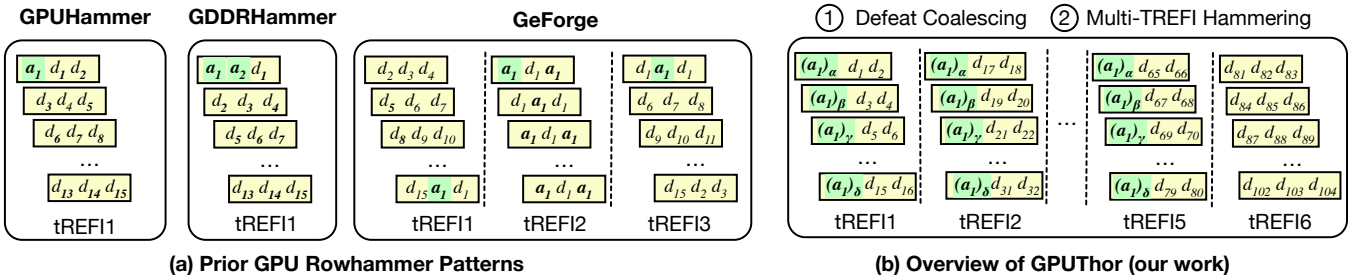

Figure 2: (a) Prior GPU Rowhammer patterns (a - aggressor row, d - decoy row, aggressor activations in green): GPUHammer and GDDRHammer hammer aggressors uniformly (1-2× activations/tREFI) with single tREFI patterns; GeForge suffers coalescing with 3 tREFI patterns. (b) Overview of GPUThor: it performs non-uniform hammering using 6 TREFI patterns with 6.6× activations per aggressor per tREFI; it avoids coalescing by repeating aggressors across warps and unique cachelines (α, β, γ, δ) .

4 Defeating GPU Memory Request Coalescing

In CPU Rowhammer attacks, hammering intensity can be easily increased by adding repeated memory accesses within a tREFI, inducing more activations. However, GPUs can perform memory request coalescing, making this challenging. While NVIDIA documentation [71] suggests that coalescing of requests occurs within warps at the L1 cache level, there is insufficient detail on request coalescing at the L2 cache or activation coalescing at the memory controller, making triggering repeated ACTs difficult. Hence, we first reverse engineer the memory request coalescing on NVIDIA GPUs and then defeat it to craft non-uniform Rowhammer patterns.

4.1 Reverse Engineering Approach

For our experiments, we reverse engineer the virtual address to DRAM bank and row mappings, similar to prior work, GPUHammer [53]. For all of our accesses, we use an `ld.volatile` followed by `discard` to ensure accesses skip the L1 and L2 caches respectively. We use an A5000 GPU for our experiments, but we validate that the memory access coalescing behavior is similar across GPUs, including A4000-A6000 (Ampere, GDDR6), L4 (Ada, GDDR6) and A30 (Ampere, HBM2).

As our baseline, we use a uniform 9-sided hammering pattern, similar to prior work GPUHammer [53], consisting of 3 Warps × 3 Threads, that accesses 9 unique rows: 1 aggressor (a) and 8 decoy rows (d_1-d_8). We then generate non-uniform patterns with 2× and 3× intensity for row a, by replacing one or two decoy rows in the pattern with repeated aggressor row (a) accesses, as shown in Figure 3. To validate whether the non-uniform patterns produce 2× or 3× activations for the repeated aggressor row (a), we measure the *time per round* for each pattern, i.e., the time to complete one round of 9 accesses in the pattern, and check if the time is consistent with 9 ACTs, similar to the baseline uniform pattern. We measure the time per round for both the uniform patterns and the non-uniform patterns, as shown in Figure 4.

To reverse engineer the coalescing behavior within and across warps, we perform the additional accesses to the row a either within the same warp as the original access, or all in different warps. We also use addresses from different cache lines, at least 128 B away (a_α, a_β, a_γ) to repeat access to row a, in same-warp and different-warp configurations, to study hammering of unique cache lines in a row.

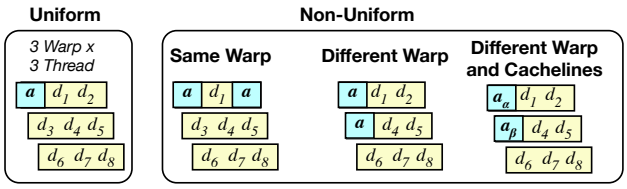

Figure 3: Non-uniform patterns (repeated accesses to aggressor rows) used to study request coalescing behavior on GPUs.

4.2 Coalescing *Within* a Warp

Figure 4 shows the time per round for the non-uniform patterns hammering additional aggressor rows, with intensity 2× and 3× within the same warp, compared to the baseline uniform hammering (1× intensity). While the uniform pattern has a time per round of 484 ns (for 9 ACTs), the non-uniform patterns with additional accesses in the same warp have a lower time per round of 469ns, indicating that the pattern did not generate 9 ACTs. This is regardless of whether the additional accesses occur to the same cache line or different cache lines in the same row. This suggests that the additional requests do not induce unique ACTs to row a, and they likely get coalesced at the memory controller.

Observation 1. Repeated requests to a row **within a warp** get coalesced at the memory controller, and **do not generate unique ACTs**, and are unsuitable for non-uniform hammering.

4.3 Repeated Requests *Across* Warps

In Figure 4, in non-uniform patterns where the aggressor row is repeatedly hammered from different warps, we see a timing spike when the repeated access is from the same cache line, jumping to 502 ns and 573 ns per round for 2× and 3× intensity hammering, from the baseline 484 ns. This is because `discard` from the first warp, which evicts the cache line from the L2 cache, interferes with the repeated request for the cache line from the second warp, which causes the latency of these accesses to increase far beyond the ACT latency. Thus, while repeated accesses across warps may not get coalesced, the timing spike due to the `discard`'s contention across

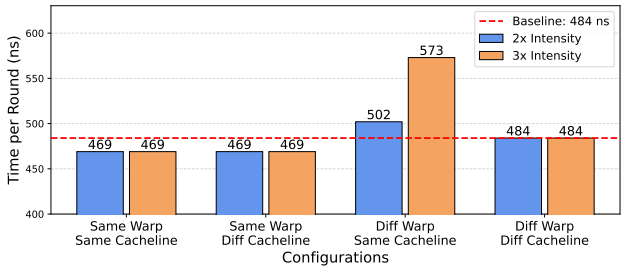

Figure 4: Average time for a round of hammering with $2-3\times$ repeated accesses to an aggressor row, accessed within or across warps, using addresses from the same or different cache lines in the row. The baseline is the time for a uniform pattern hammering 9 rows without repetition.

warps can cause the synchronization of the pattern with tREFI to be disturbed, making it unsuitable for hammering.

In contrast, when repeated accesses from different warps are to different cache lines of the row, 128 B apart, the timings are similar to those of the uniform hammering (484 ns), ensuring that there is no coalescing of requests across warps. This indicates that these non-uniform patterns likely have the same number of ACTs in total as the uniform hammering pattern, and additional ACTs for the aggressor row, a. Across different warp/thread configurations, the time per round is closest to the baseline when the repeated accesses across warps are to different cachelines, even when it does not exactly match the baseline, indicating the ACTs are preserved.

Observation 2. Repeated requests from **different warps** for **different cache lines** in a row, are not coalesced and generate **unique ACTs**, suitable for non-uniform hammering.

4.4 Non-Uniform Hammering in a Single tREFI

Using the non-uniform pattern from Section 4.3, we perform hammering campaigns. As baseline, we use a 24-sided uniform pattern with 8 warps × 3 threads, consisting of 1 aggressor and 23 decoy rows accessed uniformly within a tREFI, similar to GPUHammer [53]: this overwhelms 16-entry TRR samplers [53]. We then construct non-uniform single-tREFI patterns with 2×, 3×, and 4× intensity by repeating the aggressor access across different warps using distinct cache lines.

Table 2 reports bit flips with these non-uniform patterns on an A5000 GPU across four random DRAM banks. The 1× uniform baseline (GPUHammer) induces, on average, only 21 bit flips, while the 2× non-uniform pattern yields an average of 32.5 bit flips, a 1.5× increase. However, higher intensity patterns (3× and 4×) show a decrease in flips compared to the 2× patterns. This suggests that while non-uniform hammering with higher intensity can improve bit flips, scaling beyond 2× intensity within a single tREFI is challenging. As aggressor intensity increases, the positions of decoys become constrained within the tREFI window, reducing their chance of being sampled by TRR and their likelihood of fooling TRR. To alleviate this, we explore non-uniform patterns spanning multiple tREFIs.

Table 2: Number of bit flips on an A5000 GPU using non-uniform, single-tREFI patterns with 2×, 3× and 4× intensity, compared to the baseline 1× intensity uniform pattern [53].

	Baseline [53]	Non-Uniform, Single-tREFI		
	1×	2×	3×	4×
Bank 1	22	29	20	12
Bank 2	15	25	16	11
Bank 3	27	49	35	27
Bank 4	20	27	23	15
Average	21	32.5 (1.5×)	23.5 (1.1×)	16.3 (0.8×)

Observation 3. Non-uniform single-tREFI hammering patterns increase the number of bit flips by 1.5× by increasing intensity. But successful single-tREFI patterns are limited to 2× intensity, prompting the exploration of multi-tREFI patterns.

5 Multi-TREFI Hammering on GPUs

To overcome the limitation of single-tREFI patterns, we investigate non-uniform hammering across multiple tREFIs. Our goal is to increase aggressor intensity while continuing to evade TRR. For this, using a known bit flip in Table 2, we try to reproduce it using multi-tREFI patterns while systematically exploring the design space of non-uniform patterns. First, keeping aggressor activation counts fixed, we evaluate whether spreading them out over multiple tREFIs retains the bit flip (§5.1). Next, we replace decoys with aggressor activations to study the potential to increase intensity (§5.2).

5.1 Increasing Pattern Lengths to n tREFIs

We reverse-engineer TRR mitigation frequency using non-uniform multi-tREFI patterns by increasing pattern lengths from 1 to n tREFIs and measuring the *flip probability* for the pattern over 50 trials, defined as the percentage of trials in which the hammering pattern is able to successfully reproduce a given bit flip. As shown in Figure 5, our n-tREFI hammering pattern consists of an aggressor receiving n ACTs in a round (aggressor intensity of n) and the remaining ACTs target randomly chosen decoy rows (one ACT per decoy). We keep aggressor intensity per tREFI constant across pattern lengths, ensuring the total aggressor activations remain the same. After one round of n-tREFIs, the pattern is synchronized with the REF command via inserted delays. Since TRR sampling instances are unknown, we sweep the starting offset of aggressor activations within each round: for a round with N ACTs and aggressor intensity n, we vary the offset from 0 to $N-n$. We also study higher aggressor intensities (2× to 4× per tREFI) in these multi-tREFI patterns.

Figure 6 shows the flip probability for the same bit flip as we increase the pattern length from 1-36 tREFIs. The flip is reliably reproduced for pattern lengths of 2, 3, 4, 6, 8, 9, 12, 18, 24, and 36 tREFIs. For pattern lengths beyond 36 tREFIs, bit flips are no longer observed reliably (we observed a bit flip just once for a 48 and 72 tREFI pattern). Combined with the observation that a majority of the successful pattern lengths divide 72, this suggests that TRR mitigations likely operate every 72 tREFIs in GDDR6. Notably, higher aggressor intensities (3× and 4× per tREFI), which fail to produce

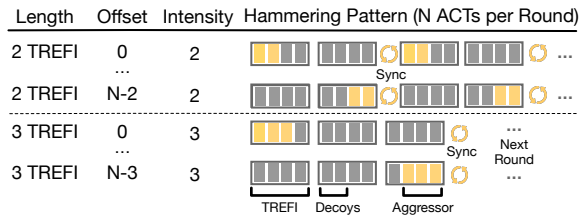

Figure 5: Hammering non-uniform patterns as the length (n-tREFIs) increases, keeping intensity per tREFI fixed.

bit flips for single-tREFI patterns, consistently reproduce bit flips across multi-tREFI pattern lengths. This shows that non-uniform multi-tREFI patterns can trigger bit flips at higher intensities.

Observation 4. Non-uniform multi-tREFI hammering reliably triggers bit flips at **higher intensities** of at least **4× per tREFI**. TRR mitigations likely apply **every 72 tREFIs**, suggesting that multi-tREFI patterns that divide 72 can reliably trigger bit flips.

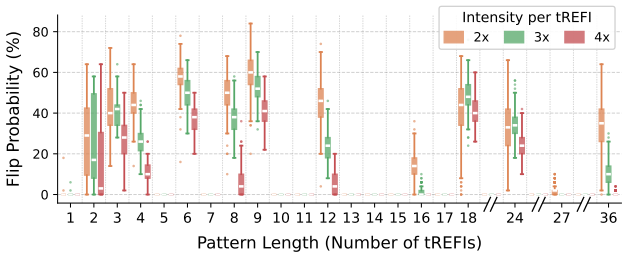

Figure 6: Bit flip probability as pattern length increases to multiple tREFIs. Bit flips are most reproducible (more than 10% flip probability) at pattern lengths of 3, 4, 6, 9, 18, and 24 tREFIs; Bit flips stop reliably appearing after 36 tREFIs.

5.2 Increasing Intensity for n tREFI Patterns

Having identified multi-tREFI patterns that reliably reproduce bit flips (patterns where lengths divide 72), we study how far we can increase aggressor intensity while maintaining reproducibility.

Approach. We start with the more reproducible multi-tREFI patterns that divide 72 tREFI (i.e., 3, 4, 6, 9, 18, and 24 tREFIs long) with a baseline aggressor intensity of 2× ACTs per tREFI, and progressively increase intensity by replacing decoy accesses with aggressor accesses. Each step increases the number of aggressor activations, while ensuring no more than one aggressor ACT per warp, to avoid coalescing as discussed in §4. Since each victim row can have two aggressor rows surrounding it, we first replace decoys greedily with just a repeated single-sided aggressor, generating patterns that increase the intensity from 1 to 7 ACTs per tREFI. Once all the warps have at least one aggressor, we follow the same procedure to add the other-side aggressor to these warps to generate patterns with intensity of 8 to 15 ACTs per tREFI. For a given pattern, after assigning aggressor activations, all the remaining activations are chosen to target unique decoy rows. For each pattern, we measure its flip probability to identify the higher-intensity patterns that reliably induce bit flips.

Results. Figure 7 shows the flip probability across 100 trials for a bit flip as the aggressor intensity increases from 2 to 15 ACTs per tREFI across the pattern, for the reliable multi-tREFI pattern lengths (e.g., lengths that divide 72 tREFI and had >10% flip probability in Figure 6). Shorter pattern lengths, such as 3 tREFI patterns, only trigger bit flips up to an intensity of 5 ACTs per tREFI. On the other hand, longer pattern lengths such as 24 tREFI patterns trigger bit flips up to an intensity of 7 ACTs per tREFI.

Across all pattern lengths, the flip probability steadily drops as ACT/tREFI increases. This is because as the number of aggressor activations increases, the number of instances when decoy rows get sampled that help to evict the aggressor from the TRR sampler decreases. Shorter patterns are able to maintain higher flip probability for bit flips at similar intensity, since shorter patterns (e.g., 6 tREFI pattern) have a higher chance of alignment with a TRR mitigation instance (every 72 tREFIs), compared to longer patterns (e.g., 24 tREFI). We desire both high intensity and flip probability to be able to trigger a larger number of bit flips in Rowhammer campaigns. Hence, for GPUThor, we choose a 6-tREFI pattern with an intensity of 6.6 ACTs per tREFI as our attack pattern.

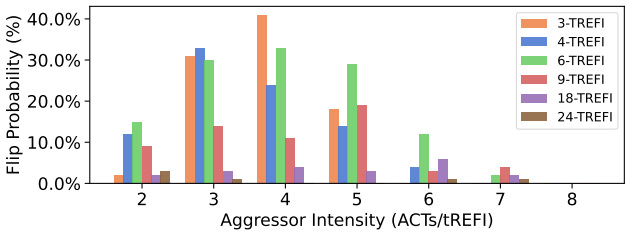

Figure 7: Multi-tREFI hammering with increasing aggressor intensity (ACTs per tREFI) for different pattern lengths (3, 4, 6, 9, 18, 24 tREFI patterns). Our patterns reproducibly trigger bit flips with intensity of up to 7 ACTs per tREFI.

Observation 5. The 6 tREFI pattern triggers flips at high aggressor intensity (6.6 ACTs per tREFI) reproducibly, making it suitable to trigger a larger number of flips in campaigns.

Single-sided vs Double-Sided. We observe that even when using two-sided aggressors in our pattern, the combined intensity of aggressors that triggers flips remains ≤ 7 ACTs per tREFI. This limit persists regardless of whether we activate only the single-sided aggressor throughout, or equally divide the activations between the two aggressor rows (double-sided). Thus, the maximum activation intensity appears to be constrained by the GDDR6 TRR implementation. A plausible explanation is that in this TRR implementation, a victim row is definitively refreshed when the combined activations from both adjacent aggressors exceed a threshold: e.g., one-third of total activations within the 72-tREFI mitigation window. Similar TRR mechanisms have been observed in other GPU memories like

HBM2 [73]. Thus, distributing activations across one or both aggressors does not increase the achievable intensity, indicating that single-sided and double-sided patterns offer similar effectiveness under our high-intensity, non-uniform, multi-tREFI hammering.

5.3 GPUThor: Putting It Together

Figure 8 shows the non-uniform, multi-tREFI pattern we use in GPUThor to enable high-intensity Rowhammer attacks on GPUs. The non-uniform hammering uses repeated aggressor activations in distinct warps and uses distinct cache lines within a tREFI to defeat request coalescing. The pattern spans 6 tREFIs, using an aggressor intensity of 6.6 ACTs/tREFI across the entire pattern. The first 5 tREFIs contain repeated aggressor activations (one per warp) and decoy activations, and the last tREFI only contains unique decoy row activations. We observe GPUThor achieves up to 110K aggressor ACTs per tREFW, which is 6.6× more than prior uniform hammering patterns with GPUHammer [53].

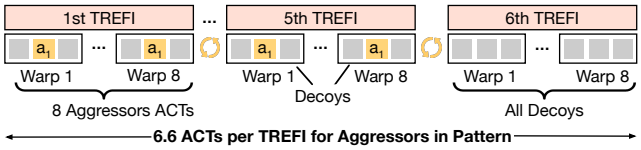

Figure 8: GPUThor Hammering Pattern.

6 Results

We evaluate GPUThor on a broad set of NVIDIA GPUs with GDDR6 memories to study its bit-flip behaviors and real-world implications. Our evaluations focus on answering the following questions:

(1) Does GPUThor induce significantly more bit flips than prior attacks on a wide range of NVIDIA GPUs? (§6.2)
(2) Do the increased bit flips with GPUThor reduce the time needed for GPU Rowhammer-based privilege-escalation exploits? (§6.3)
(3) Does GPUThor induce multi-bit flips that can potentially defeat ECC on GDDR6 based NVIDIA GPUs? (§6.4)
(4) Can GPUThor trigger exploits on GPUs despite NVIDIA-recommended mitigations like ECC being enabled? (§7)

6.1 Experimental Setup

Target GPUs. We evaluate GPUThor across four Ampere-class NVIDIA GPUs: the RTX A4000, A4500, A5000, and A6000, spanning the GA102 and GA104 architectures with 16–48 GB GDDR6 memory. The RTX A6000 is hosted locally, while the A4000, A4500, and A5000 are accessed on the cloud. Our A6000 uses Samsung GDDR6, as confirmed using `LACT` [32]; the cloud GPUs do not expose the low-level permissions required to identify the memory vendor. All experiments in §6 are conducted with ECC disabled, reflecting the default in several cloud GPUs (our cloud GPUs do not enable ECC by default, nor provide us the permissions to enable ECC); we enable ECC for the local A6000 for evaluations in Section 7. We use Ubuntu 22.04 with IOMMU enabled, like prior work [55].

Recovering Address-to-Row Mappings. Like GPUHammer [53], we reverse-engineer the GPU virtual-address-to-DRAM bank and row mapping in two steps. First, we use timing side-channels on row-buffer conflicts [53, 77] to recover addresses that map to the same bank. Within each bank, we identify addresses mapping to the same row based on row-buffer-hit timing and identify unique addresses mapping to the same bank and consecutive rows. We use this set of addresses, called the *row set*, for hammering campaigns.

Hammering Configuration. Our hammering kernels use 8 warps × 3 threads, with each thread hammering one row in a tREFI like GPUHammer [53]; the pattern synchronizes with REF every n−tREFI through added delays. We introduce an outer loop for multi-tREFI patterns to hammer a different schedule of activations every tREFI for GPUThor's non-uniform patterns. Our hammering campaigns sequentially select each row in the bank as aggressor, and hammer using four victim/attacker data patterns: 0xff/00, 0x00/ff, 0xaa/55, 0x55/aa. Our hammering campaigns last for 24 hours/bank on each GPU and we hammer four banks per GPU.

Exploits. As a proof-of-concept for exploitation with our bit flips, we use the GPU page-table corruption exploit similar to prior works [31, 55, 102]. For this, we use the open-sourced GPUBreach exploit code [54]. Here, the attacker massages PTEs into target DRAM rows, hammers neighboring aggressors to flip the page-frame-number (PFN) bits of a PTE, and ensures the tampered PTE maps to another PTE via an additional massaging step, to gain arbitrary read/write privileges to the entire memory. For the ECC-enabled settings, we describe our exploit methodology in §8.

6.2 Bit Flip Characterization

Table 3 shows the results of our hammering campaigns on 4 GPUs (A4000-A6000) targeting 4 DRAM banks each. While GPUHammer [53] produces 19 to 658 bit flips on these GPUs, GPUThor yields 18,000 to 94,000 unique bit flips. GPUThor achieves 150× (A5000) to 2,602× (A6000) more bit flips compared to GPUHammer.

Across the four GPUs, we observe that the A5000 GPU has the highest vulnerability levels, resulting in 377K bit flips per GB. The A6000 GPU has the second-highest vulnerability level, with 114K bit flips per GB. The A4500 and A4000 have the lowest vulnerability levels with 75K and 72K flips per GB. The bit flip rates with the GDDR6 DRAM on the A5000 GPU are close to the worst-case vulnerability in DDR4 DRAM with state-of-the-art CPU attacks like Blacksmith (550K flips per GB) [35], indicating that our non-uniform GPU Rowhammer attacks produce similar worst-case bit flip rates as the best CPU Rowhammer attacks.

On the A5000, the most vulnerable GPU, we observe 3,584 flips per hour (one per second) for the best data pattern with GPUThor. This rate, achieved by hammering a single bank at a time, is almost 180× faster than the prior best of 20 flips per hour with GDDRHammer [31] and multi-bank hammering (6 banks hammered in parallel). Across GDDR6 banks of a GPU, the vulnerability levels are similar: e.g., on the A5000, the average (23597 flips per bank) is similar to that of the most vulnerable bank (31797 flips). Across the GPUs, we observe an average of 0.15-0.76 flips per row. This makes multiple flips per row and cache line a common occurrence, and multi-bit flips capable of defeating ECC likely (§6.4).

Across the GPUs, we also measure the minimum activation count to trigger a bit flip (HC_{first}), by re-hammering the bit flips from GPUHammer patterns with activation counts decremented by 100

Table 3: Rowhammer bit flips from campaigns on A4000, A4500, A5000 and A6000, hammering 4 banks each (24 hours / bank). Our campaigns use GPUThor and prior work, GPUHammer [53].

GPUs				GPUHammer [53]	GPUThor (Our Work)					
Name	Arch	VRAM	Rows / Bank	Total Unique Flips	Total Unique Flips	Avg. Flip / Row	Avg. Flip / Bank	Avg. Flip / GB	Flip / Hour (Best Data Pattern)	Min. HC_{first}
A6000	GA102	48GB GDDR6	64K	22	57247 (**2602**×)	0.22	14311	114K	1501	12.3K
A5000	GA102	24GB GDDR6	32K	658	94389 (**150**×)	0.76	23597	377K	3584	15.8K
A4500	GA102	20GB GDDR6	32K	19	18756 (**987**×)	0.15	4689	75K	882	13.6K
A4000	GA104	16GB GDDR6	32K	23	18192 (**790**×)	0.15	4548	72K	838	12.4K

Table 4: Time to mount privilege-escalation exploit using GPUThor compared to GPUHammer [53].

GPU	GPUHammer [53]				GPUThor (our work)			
	Row Set (min)	Bit-Flip Discovery (min)	Privilege Escalation (min)	Total (min)	Row Set (min)	Bit-Flip Discovery (min)	Privilege Escalation (min)	Total (min)
A6000	90	1223.6	0.3	1313.9	0.5	0.3	0.3	1.1
A5000	3.1	3.6	0.2	6.9	0.3	0.1	0.2	0.7
A4500	8.8	237.5	0.2	246.6	0.5	0.5	0.2	1.2
A4000	10	403.8	0.2	414.0	0.4	0.6	0.2	1.2

until the flips stop appearing. All of the GPUs have HC_{first} values between 12.3K and 15.8K, with no direct correlation between the GPU's overall vulnerability and its HC_{first} value.

6.3 Time to Exploit

As GPUThor discovers bit flips significantly faster than prior work [53], it directly reduces end-to-end exploit time. We demonstrate this using a privilege-escalation exploit that corrupts GPU PTEs via Rowhammer, following prior attacks [31, 55, 102].

Setup. We implement a privilege escalation exploit, similar to GPUBreach [55], using their exploit code [54], replacing the default uniform hammering kernel with GPUThor's non-uniform hammering. We evaluate the exploit using both GPUHammer [53] and GPUThor hammering, and measure the total time including the *offline* and *online* phases of the exploit. The *offline* phase includes the discovery of the *row-set* (addresses mapping to rows of the same bank), and the hammering campaign to find exploitable bit flips; the *online* phase includes massaging the PTE entries to vulnerable rows and hammering the neighbors to escalate privileges [55].

Results. Table 4 shows the total exploit time across four GPUs. With GPUHammer, the end-to-end exploit can take up to several hours (21.9 hours on A6000 and 7 hours on A4000). The A5000, the most vulnerable GPU, is the exception where the exploit succeeds in 6.9 minutes. The main bottleneck in the exploit is the *offline phase*, particularly the exploitable bit-flip discovery, which takes almost 20.3 and 6.7 hours on the A6000 and A4000. This is because for the exploit to succeed [55], we need the bit flips to be at specific offsets (25 bits of the page frame number in the 8 B PTE entry). This takes only 3.6 minutes on the A5000, the most vulnerable GPU.

In contrast, GPUThor reduces bit-flip discovery time to less than a minute (0.1–0.6 minutes), bringing the total exploit time down to 0.7–1.2 minutes across all the GPUs (10× to 1200× lower than GPUHammer). This is due to the significantly higher rate of exploitable bit flips discovered by GPUThor. This also makes row-set discovery time faster, as GPUThor requires a smaller row-set (tens of rows) given the higher spatial density of its discovered bit flips, compared to GPUHammer that requires larger row sets (tens of thousands of rows across multiple banks).

The online phase (PTE massaging and final hammering) remains unchanged in GPUHammer and GPUThor (0.2–0.3 minutes) and is a small fraction of the total runtime.

6.4 Multi-Bit Flips

Beyond increasing the number of bit flips compared to prior works [31, 53, 55, 102], GPUThor also provides bit flips with higher spatial density. Table 5 quantifies the number of multi-bit flips observed in GPUThor's hammering campaigns across GPUs at 8 B, 16 B and 32 B granularity. These are based on the potential ECC code granularity in GDDR GPUs that store 2 B ECC per 32 B data [94].

At 16 B and 32 B granularities, we observe **double-bit flips** on all GPUs, with counts increasing at larger granularities. This is most pronounced on the A5000 and A6000, with up to 137 and 26 double-bit flips per bank at 16 B granularity (276 and 54 within 32 B), respectively. The A4000 and A4500 exhibit fewer multi-bit flips, consistent with their lower overall vulnerability. These double-bit flips can potentially result in detectable, uncorrectable errors (DUE) with SECDED ECC, causing data loss or denial of service.

The A5000 also exhibits two **triple-bit flips** at a 16 B granularity and four at a 32 B granularity (two of them also fall within an 8 B region). These are particularly concerning, as they exceed both the correction and detection capability of SECDED ECC on GDDR-based GPUs, resulting in silent data corruption (SDC). Next, we evaluate implications of GPUThor's bit flips on ECC-enabled GPUs.

7 Implications for ECC Protections on GPUs

As GPUThor can induce multi-bit flips beyond ECC's correction capability, we analyze how NVIDIA's ECC behaves under multi-bit

Table 5: Multi-bit flips across A4000, A4500, A5000, A6000 at 8-, 16- and 32-byte granularity. All the GPUs have *double* bit flips, while the A5000 also has *triple* bit flips.

	Double Bit Flips								Triple Bit Flips		
	A4000		A4500		A5000		A6000		A5000		
	16B	32B	16B	32B	16B	32B	16B	32B	8B	16B	32B
Bank 1	6	13	5	6	73	138	26	54	1	1	2
Bank 2	2	6	4	11	40	79	15	27	-	-	-
Bank 3	4	10	-	-	137	276	7	11	1	1	2
Bank 4	1	3	6	10	56	128	5	10	-	-	-

flips by running Rowhammer campaigns using GPUThor with ECC enabled (§7.1), derive insights on NVIDIA's ECC algorithm (§7.2) and discuss potential exploits on ECC-protected GPUs (§7.3).

7.1 Reverse Engineering ECC on NVIDIA GPUs

The exact ECC algorithm used by NVIDIA is undisclosed. However, prior work [94] has speculated that GDDR6 NVIDIA GPUs provision 2B of sideband ECC per 32B data access, resulting in a 6.25% memory storage overhead. Still, reverse engineering ECC on GDDR memories is challenging since we lack logic analyzers or FPGA-based testbeds for GPU memories that enabled reverse engineering of ECC on CPU DRAM [8, 42]. Thus, we hammer blindly with ECC enabled and leverage side-channels to infer details about the ECC scheme.

Approach. We enable ECC on our GPUs and execute hammering campaigns with GPUThor. In the event of correctable errors, NVIDIA GPUs do not provide logs specifying the bit locations of corrected flips. Moreover, we observe that NVIDIA GPUs do not exhibit any timing variations on memory reads when the ECC corrects a bit flip, unlike similar timing side-channels on CPUs observed in prior works [8, 42]. Instead, we discover two *new* side channels: (1) the number of corrected errors reported by `nvidia-smi` in the "`Volatile DRAM Correctable`" counter, and (2) a timing variation in the completion time of the GPU kernel, due to the delivery of these ECC counter updates to the CPU driver. As shown in Figure 9, the kernel completion time shows an increase from 4.5ms→7.6ms when an error is corrected during the kernel execution. These kernel execution times are much less than the latency to probe `nvidia-smi` (178 ms); hence, we use the kernel completion-time side-channel in our campaigns to detect correctable errors.

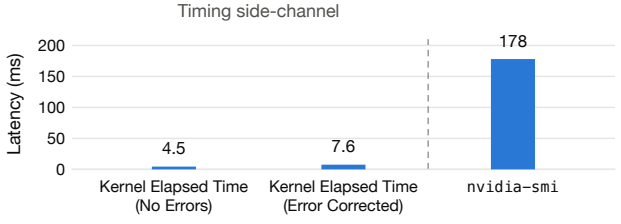

Figure 9: Side channels allowing detection of corrected errors: kernel completion time increases when an error is corrected during execution (4.5ms → 7.6ms); nvidia-smi that also reports corrected error counts has higher latency (178 ms).

In the event of detectable but uncorrectable errors (DUE), the CUDA kernel crashes and subsequently requires a GPU reset: fortunately, the GPU driver logs the byte location of such errors to `syslog`, allowing us to infer their locations. In such events, we use a GPU reset to restart our campaigns. DUEs also trigger a row-remapping on Ampere GPUs [69], which remaps the row where the DUE is detected to a spare row in the bank (up to 8 remaps per bank permitted). In our campaigns, we flash the GPU InfoRom with `nvflash` [98] before a GPU reset to skip the row-remapping. We discuss later (§8) how an unprivileged attacker can work around row-remapping for exploits with uncorrectable bit flips.

When we detect a correctable error or DUE, to recover the exact *bit* location within a row of the bit flip, we apply *masking* [8, 12, 42] similar to prior works. Here, we invert a selected bit in the row in the expected direction of a flip, and try to re-trigger the error by hammering; if it fails to trigger, then we know the bit flip location.

If a triple bit flip is triggered, it surpasses the ECC's detection capabilities and leads to a Silent Data Corruption (SDC); here we observe multiple bits remain flipped even after our side-channel indicates an error was corrected and the kernel does not crash.

Campaign Setup. The A5000 GPU, that is the most vulnerable in our test set, is cloud-based (as are the A4000 and A4500), where the cloud provider does not expose permissions to enable ECC. Thus, we only run campaigns with ECC enabled on the local A6000 GPU. We run a campaign on one bank, where hammering blindly with ECC enabled to find DUEs takes ~1 day for all data patterns, and a campaign that also identifies bit locations of flips in each DUE and corrected SBE via bit masking takes ~1.5 days per data pattern.

7.2 Insights on NVIDIA ECC Implementation

In our campaign with ECC enabled on the A6000, we observe 11 DUEs and 1 SDC on a single bank. We derive several insights about the NVIDIA ECC based on this.

ECC Granularity. Across the campaign, all the DUEs have at least two bits flipped within a 32B granularity, indicating that NVIDIA's ECC code is calculated at the granularity of ≤32B. Additionally, we observe some instances of two bit flips within 32 B data that are both corrected simultaneously, i.e., one bit flip per 16 B on average that is correctable. This suggests that NVIDIA uses a SECDED ECC scheme where 1 B ECC code protects 16 B of data. Given that each 32 B data stores a 2 B ECC, there are two 16 B non-contiguous chunks of data, each protected by a 1 B SECDED ECC.

Observation 6. NVIDIA ECC protects 32B data with a 2×1 B SECDED code, where each 1B ECC provides SECDED protection for 16 B of data (non-contiguous halves of 32B data).

Miscorrection on DUEs. On a DUE, along with the two bits that flip due to Rowhammer (identified by masking those bits and trying to re-trigger them), we observe additional bits flipping within 32 B data, indicating the ECC incorrectly attempts correction and miscorrects the data even on a DUE. Across 11 DUEs observed in the hammered bank, each has 2-10 flips (5 on average). The extra miscorrection flips on DUE are independent of the data value (0 or 1) and always occur at the same locations relative to the two

rowhammer-induced flips. Typically, the original Rowhammer bit flips are preserved in the miscorrected data.

Delayed DUE Handling by Driver. Most importantly, on a DUE, the miscorrected data *remains accessible* from the GPU kernel for up to **10 ms**. Although no new CUDA API calls can be triggered from the CPU, the corrupted data can be consumed by the CUDA kernel on the GPU and used for page table translations if a GPU PTE is corrupted. We use this insight for our DUE-based privilege escalation exploit in §8. The GPU context remains accessible to CUDA kernels for ~10 ms after the DUE, after which all the kernels are terminated, and the GPU is unusable until a full reset.

Observation 7. After a DUE, GPU kernels remain **alive for 10 ms** and can **consume the corrupted data** or illegally **access data outside their process** through corrupted PTEs.

Triple-Bit Error inducing SDC. While a two-bit flip yields a DUE, we also found a *triple-bit* flip within the ECC granularity that gets mis-corrected without crashing the kernel, resulting in an SDC. A three-bit error is known to exceed the detection *and* correction capacity of a SECDED-based Hamming code, so its syndrome aliases to that of a single-bit error [27]. We observe NVIDIA ECC exhibits similar behavior, where it flags a triple-bit flip as a *correctable* error, mis-corrects it to end up flipping a fourth bit whose location is fixed deterministically based on the location of the other three flips. Since the triple-bit flip miscorrections do not crash the kernel, these enable privilege escalation exploits [55], as shown in §8.

Observation 8. A *triple-bit* flip within the ECC granularity can induce **Silent Data Corruption** under NVIDIA's ECC.

7.3 Exploits on ECC-Enabled GPUs

Denial of Service (DoS) Exploits. All our GPUs belong to the GA10X architecture, which does not support Error Containment [69]. Thus, a single DUE kills all running processes on the GPU, requiring a full GPU reset to recover, resulting in a DoS. On the A6000 with ECC enabled, we can trigger 11 DUEs on a single bank within one day of hammering (1 DUE every ~2 hours). Since GPUs in the cloud have been shown to have a Mean Time To Repair (MTTR) of 0.3 hours [9] to reset the entire node, this reduces the GPU node's availability under a DoS attack by 13% (3.3 hours of downtime per day) and causes data loss due to frequent crashes.

Abusing Return Merchandise Authorization (RMA) Policy. NVIDIA GPUs have an RMA policy [69] for defective products. A GPU becomes RMA eligible when row-remapping fails: each GPU bank has 8 spare rows and each DUE triggers a row remap. If any bank experiences more than 8 DUEs, a row-remapping failure flag is set (visible via `nvidia-smi`), making the GPU eligible for RMA. From Table 5, 2 of our 4 GPUs meet this condition, indicating that GPUThor can be used by malicious customers to trigger RMA eligibility and request a free GPU replacement. In our ECC-enabled campaign, the A6000 has the row-remapping failure flag set after the 9th DUE, within 18 hours. This poses a risk of warranty abuse by GPU owners, with potential financial implications for GPU vendors.

Privilege Escalation under ECC Miscorrection. Observation 7 and Observation 8 suggest that ECC miscorrections on DUEs and SDCs can both be exploited for potential privilege escalation attacks. We discuss this in more detail in §8.

8 Privilege Escalation Exploit with ECC Enabled

8.1 Overview

We demonstrate a privilege-escalation exploit on ECC-enabled GPUs, building on prior exploits [31, 55, 102] through Observation 7 and Observation 8. Prior attacks [31, 55, 102] follow three key steps: (1) massage page tables (PTs) to vulnerable rows, (2) induce bit flips by hammering neighbors, and (3) massage a new PT at the destination of the corrupted PT entries to control GPU page tables and escalate privileges by accessing and tampering with privileged CPU memory. However, with ECC enabled, step (3) *can* be infeasible as GPU memory allocations, needed for massaging GPU PTs, are not permitted after a flip in the case of a DUE. Thus, for our exploit, we pre-place a PT into a location *predicted* to be the destination of a corrupted PT (Figure 10). After a successful PT corruption, this allows us to (i) access and modify the pre-placed PTE, altering it to point to CPU memory using the Aperture bits ("A" in Figure 11) and (ii) use it to tamper with privileged CPU memory (within 10 ms before the GPU kernel terminates in the case of a DUE), enabling privilege escalation. Our threat model assumes a single-tenant setting for the A6000 with ECC enabled.

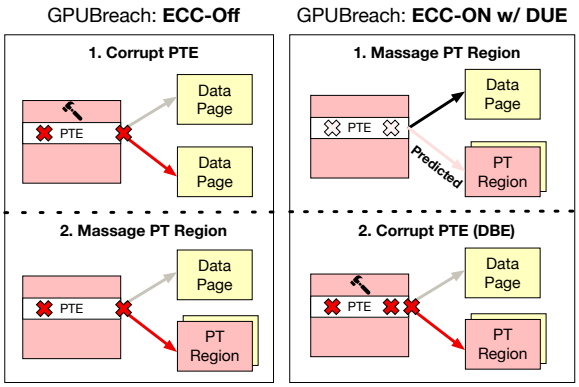

Figure 10: Privilege escalation exploit with GPUBreach [55] when ECC is enabled using DUEs. We predict the corrupted PTE value and massage a PT into the predicted location in advance, enabling the exploit within 10 ms of a DUE.

8.2 Offline Profiling for DUEs and SDCs

We target 2 MB PTEs for corruption with bit flips (Figure 11). In these PTEs, a multi-bit flip is exploitable only if (1) some of its bit flips reside in the Page Frame Number (PFN) region of the PTE (25 out of 128 bits) and (2) the tampered PFN ends up in user-accessible memory. An additional challenge is that miscorrections from either a DUE or an SDC can flip additional bits along with the Rowhammer-induced errors. While the original flip locations can be identified easily (Section 7.1), the miscorrections on DUEs

are hard to predict. Fortunately, out of the 11 DUEs we found in Section 7.2, 3 have flips within the PFN suitable for our exploit. The discovered SDC also has exploitable flips within the PFN.

Figure 11: 16B PTE Format for 2MB Page Frames. [68]

To practically develop an exploit, we need to overcome the remapping of rows on DUEs or induce an exploitable DUE or SDC without triggering the row-remapper. We develop two approaches for this.

Approach 1: Exhaust Row-Remapper. After 8 DUEs on a bank, the GPU no longer remaps rows and triggers the remap failure flag on the 9th DUE. Subsequently, any DUEs in the bank persist and become reproducible even after GPU resets and system reboots. Thus, the location of the DUE and the destination of corrupted bit flips in PTEs (i.e., bit flip locations) can be precisely identified. We can trigger this state within 18 hours of hammering on our A6000.

Approach 2: Exhaustive Bit-Search. Similar to prior ECC-based exploits [8, 42], we identify the two or three constituent bit flips of a double or triple bit flip, without triggering them together to avoid a DUE that would trigger a row remap. To that end, we use the insight from SoftHammer[13], where they show the attacker can gradually increase hammering intensity to avoid collateral damage. Our bit search runs as follows: first, for each row, we gradually increase intensity ($2\times \rightarrow 3\times \rightarrow \ldots$) until we observe the first flip in a row. Then, we employ the bit-by-bit search [42] within each 32B segment of the row, masking each of the 256 bits, to find the second or the third flip without triggering the prior flips. This process to locate exploitable double and triple bit errors, without triggering DUEs, takes ~4 days on our A6000.

8.3 Online Phase: PT Tampering

Prior work [31, 53, 55, 102, 120] shows that GPU physical memory allocation is contiguous and reproducible in single-tenant settings. This allows us to deterministically predict the page frame a PTE will reference after tampering, given the multi-bit flip locations.

Massaging Page Tables. We leverage `cuMemMap` [31, 55] to duplicate PTEs with identical physical page frame numbers within a PT region. Given the contiguous virtual-to-physical mappings, the corrupted PTE's destination corresponds to the original virtual address plus the known bit-flip location offset. In advance of the hammering, we (1) massage a PT region into the vulnerable row, (2) fill it with duplicated PTEs with `cuMemMap`, (3) massage a second PT region at the predicted tampered PTE's destination and fill it using `cuMemMap`.

Obtaining Arbitrary Read/Write. With the PTEs in place, we trigger the profiled double-bit flip in the target PTE using GPUThor. Then we launch a second GPU kernel that accesses the virtual address obtained from the first `cuMemMap`, corresponding to the tampered PTE. If the PTE tampering is successful, we can access the PTEs of the second PT region, and modify them to read/write arbitrary GPU memory or privileged host CPU memory (setting the aperture bits in the PTE [31, 55, 102]).

8.4 Results

On the ECC-enabled A6000 GPU, we found one SDC and 3 out of 11 DUEs suitable for exploitation (§8.2). For the offline phase, we use Approach-2 (§8.2), which completes in 4 days. The online phase takes under 2 minutes to obtain an arbitrary read/write primitive, with `cuMemMap` and TLB flushing accounting for most of that time.

SDC-based exploit. Using the SDC, the privilege escalation to root on the host is straightforward, as the program does not crash after obtaining the arbitrary Read/Write primitive on the GPU memory. By directly leveraging prior work [55], we achieve host-side privilege escalation with ECC enabled on the GPU, even when the IOMMU is enabled.

DUE-based exploit. Using DUEs, once the exploit performs the memory accesses using corrupted PTEs to achieve the arbitrary Read-Write privileges, the GPU kernel terminates within 10 ms. Given this limited time window, this makes directly applying the GPUBreach [55] exploit challenging. On systems where the IOMMU is disabled by default, a common occurrence in Ubuntu [63, 102], we can still achieve host-side privilege escalation within this brief time window, following prior works [31, 102]. Specifically, we modify the pre-placed PTE's aperture bits ("A" in Figure 11), to map GPU accesses to CPU memory, and use the arbitrary write primitive to overwrite the process's credential structure (`cred`), whose address is assumed to be obtained via side channels [10, 55, 56]. By setting `euid=0`, we get root privileges on the host, similar to prior works [31, 102]; this persists even after the GPU kernel crashes. These steps complete in under 1 ms, within the available time window, practically achieving host-side privilege escalation even with ECC enabled on the GPU, on host systems with IOMMU disabled.

9 Discussion and Limitations

Applicability. Our high-intensity patterns developed in GPUThor provide bit flips on every GA10x GPU with GDDR6 memory we tested (A4000, A4500, A5000, A6000). We also tested other memory types (see Appendix D), including HBM, GDDR6X, and newer generation GDDR6 on NVIDIA GPUs, and did not observe any bit flips on them. This is likely due to differing TRR implementations in these memories compared to the A4000-A6000 GPUs, which make GPUThor's patterns unsuccessful. Some GDDR6X-based chips also deploy Refresh Management (RFM) [67], further complicating Rowhammer attacks. Nevertheless, we demonstrate the feasibility of non-coalescing primitives enabling non-uniform, multi-tREFI hammering on NVIDIA GPUs; this enables future work to systematically explore attack patterns on these additional GPUs.

Newer Reliability Mechanisms in GPUs. Newer NVIDIA GPUs support additional reliability mechanisms [69] with implications for some of our exploits. On server-class Ampere GPUs (A100) and beyond, *Error Containment* and *Dynamic Page Offlining* isolate faults to the triggering applications, avoiding full GPU resets or crashes of co-tenant processes, improving resilience to denial-of-service attacks. However, these still rely on SECDED-level ECC; if this is entirely bypassed with SDCs, i.e., triple-bit flips [8, 42], privilege escalation attacks can still work. *RAS Repair*, introduced in some Blackwell GPUs, replaces a DRAM channel after repeated row-remapping failures within a bank. While this makes the attack using DUEs more time-consuming, since it is primarily an extension of

row remapping, it does not prevent the attack. Finally, newer GPUs adopt HBM3/e and GDDR7 with on-die ECC [38, 40]. While this reduces error visibility, it remains vulnerable if multi-bit flips get triggered [65]. Future work can explore exploits on these platforms.

10 Mitigations

Stronger Error Correction. Deploying stronger ECC [26, 66] like Chipkill [14] can improve resilience to multi-bit flips and enhance Rowhammer protections. However, increasing ECC strength, either on-die or via side-band, may incur non-negligible storage and bandwidth overheads, which can be costly in GPU memory systems [53, 94, 114]: existing SECDED-level ECC in GDDR6 already introduces a 6.25% memory overhead and up to 10% slowdown. This motivates future research on efficient error correction schemes [46, 60, 78, 107, 112] that can offer both stronger protections against multi-bit flips and incur low overheads.

ECC State Monitoring. Just as an attacker can use `nvida-smi` reported correctable error counts to identify bit flips (Section 7.1), an administrator can monitor the ECC-related state to detect Rowhammer activity. A spike in row-remapper activity or number of corrected errors can reveal an attack. However, such monitoring also has two limitations that may enable evasion: (1) the row remapper is triggered only by DUEs, not SDCs, and (2) the correctable-error counter becomes unreliable after a few thousand errors and then largely stops incrementing. We leave a detailed investigation of such monitoring mechanisms for future work.

Principled Hardware-Level Defenses. Adopting more advanced in-DRAM mitigations, such as Refresh Management (RFM), proposed for HBM3 [38], GDDR6/X, GDDR7 [37, 40], and DDR5 [34, 39] can make attacks like GPUThor harder. Per-Row Activation Counting (PRAC) proposed in the DDR5 specification [1, 6, 80, 108, 109] provides a more principled defense that eliminates the vulnerability almost entirely. DRAM integrity protections [18, 41, 88] can guarantee the detection of data corruption via Rowhammer and thus prevent exploits. Furthermore, adopting memory-controller-based defenses [79, 82, 86, 89, 96, 110, 111], or leveraging address scrambling techniques [87, 107] can also reliably thwart attacks.

11 Related Work

Rowhammer Attacks. The Rowhammer phenomenon has been extensively studied on DDRx and LPDDRx DRAM in CPU platforms [44, 51, 59, 72, 75, 101, 122]. TRRespass [21], SMASH [11], Blacksmith [35], ZenHammer [36], and Phoenix [65] develop attack patterns that defeat in-DRAM mitigations from DDR3–5. Other attacks [20, 48] have been shown on LPDDRx DRAM. ECCploit [8] and ECC.fail [42] show that modern ECC memories do not stop Rowhammer. Exploits demonstrated using Rowhammer include privilege escalations [91], cryptographic fault injections [85], RDMA-based attacks [97], browser compromises [11, 12, 23], and attacks on ML models/libraries [7, 16, 30, 52, 84, 92, 115]. Recent works demonstrate variants of the read disturbance phenomenon like RowPress [58] and ColumnDisturb [118]. GPUHammer [53] first demonstrated Rowhammer on discrete GPUs. Subsequent work attacked GPU page tables resulting in privilege escalation [31, 55, 102]. GPUThor translates non-uniform hammering popular on CPUs [35] to GPUs, enabling the first high-intensity attacks on GPUs, compromising their ECC protections.

GPU Vulnerabilities. Prior work exposes a broad attack surface on GPUs spanning data leakage, side channels, and memory corruption. Residual GPU memory can leak renders and ML outputs [50, 93, 123]. Side-channel attacks can leak pixels [103, 104, 119] and extract ML model parameters [17, 105]. Covert channels exploiting microarchitectural leakage have also been demonstrated [67, 120]. In parallel, memory safety bugs (e.g., buffer overflows) enable control-flow hijacking and ML model tampering [25]. More recently, CPU-GPU communication channels (e.g., RPC queues) have been shown to leak sensitive data even in confidential computing settings [24]. GPU Rowhammer [31, 53, 55, 102] expands this surface, enabling privilege escalation. GPUThor extends this line of work by demonstrating GPU Rowhammer exploits with ECC enabled.

Rowhammer Mitigations. Most defenses target CPU DRAM. Software approaches focus on memory isolation [2, 4, 49, 57, 90, 100], ECC-driven remapping [15, 69], or aggressor rate limiting [5, 19, 116]; these may extend to GPUs with driver support. Hardware-level defenses, such as tracker-based schemes [3, 33, 45, 61, 74, 76, 79, 81, 82, 96, 117, 121], row relocation [86, 89, 106, 111], delayed activations [116], refresh-generating activations [62], page table integrity protection [88], and PRAC-based designs [6, 43, 64, 80, 83, 95, 109, 113], are also applicable to GPU DRAM. However, their latency, bandwidth, and storage overheads on GPUs remain unexplored.

12 Conclusion

We present GPUThor, the first high-intensity, non-uniform Rowhammer attack on NVIDIA GPUs. By reverse-engineering coalescing on GPUs and TRR on GDDR6 memories, we design multi-tREFI patterns that achieve 500×–23,500× more bit flips than prior GPU attacks. GPUThor also induces multi-bit flips uncorrectable by the SECDED ECC, and enables denial-of-service attacks and the first privilege-escalation attacks on ECC-enabled GPUs. Our results show that the NVIDIA-recommended mitigation of enabling ECC in GPUs is insufficient and stronger defenses are needed.

Acknowledgments

We thank Prof. Kaveh Razavi for a conversation that inspired this study. This research was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants under funding reference number RGPIN-2023-04796, and an NSERC-CSE Research Communities Grant under funding reference number ALLRP-588144-23. Any research, opinions, or positions expressed in this work are solely those of the authors and do not represent the official views of NSERC, the Communications Security Establishment Canada, or the Government of Canada.

References

[1] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. 2021. Panopticon: A complete in-dram rowhammer mitigation. In *Workshop on DRAM Security (DRAMSec)*, Vol. 22. 110.

[2] Carsten Bock, Ferdinand Brasser, David Gens, Christopher Liebchen, and Ahamd-Reza Sadeghi. 2019. RIP-RH: Preventing rowhammer-based inter-process attacks. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 561–572.

[3] F. Nisa Bostanci, ISmail Emir Yüksel, Ataberk Olgun, Konstantinos Kanellopoulos, Yahya Can Tuğrul, A. Giray Yağlıçi, Mohammad Sadrosadati, and Onur

Mutlu. 2024. CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 593–612. doi:10.1109/HPCA57654.2024.00050

[4] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. CAn't touch this: Software-only mitigation against Rowhammer attacks targeting kernel memory. In *26th USENIX Security Symposium (USENIX Security 17)*. 117–130.

[5] Oğuzhan Canpolat, A. Giray Yağlıkçı, Ataberk Olgun, Ismail Emir Yuksel, Yahya Can Tuğrul, Konstantinos Kanellopoulos, Oğuz Ergin, and Onur Mutlu. 2024. BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads. In *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture* (Austin, TX, USA) *(MICRO '24)*. IEEE Press, 915–934. doi:10.1109/MICRO61859.2024.00072

[6] Oğuzhan Canpolat, A Giray Yağlıkçı, Geraldo F Oliveira, Ataberk Olgun, Nisa Bostancı, Ismail Emir Yuksel, Haocong Luo, Oğuz Ergin, and Onur Mutlu. 2025. Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[7] Zachary Coalson, Jeonghyun Woo, Chris S. Lin, Joyce Qu, Yu Sun, Shiyang Chen, Lishan Yang, Gururaj Saileshwar, Prashant Nair, Bo Fang, and Sanghyun Hong. 2025. PrisonBreak: Jailbreaking Large Language Models with at Most Twenty-Five Targeted Bit-flips. *arXiv preprint arXiv:2412.07192* (2025).

[8] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. 55–71. doi:10.1109/SP.2019.00089

[9] Shengkun Cui, Archit Patke, Hung Nguyen, Aditya Ranjan, Ziheng Chen, Phuong Cao, Gregory Bauer, Brett Bode, Catello Di Martino, Saurabh Jha, Chandra Narayanaswami, Daby Sow, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2025. Story of Two GPUs: Characterizing the Resilience of Hopper H100 and Ampere A100 GPUs. arXiv:2503.11901 [cs.DC] doi:10.1145/3712285.3759821

[10] Davide Davoli, Martin Avanzini, and Tamara Rezk. 2024. On Kernel's Safety in the Spectre Era (And KASLR is Formally Dead). In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1091–1105.

[11] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2021. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1001–1018. https://www.usenix.org/conference/usenixsecurity21/presentation/ridder

[12] Finn de Ridder, Patrick Jattke, and Kaveh Razavi. 2025. Posthammer: pervasive browser-based rowhammer attacks with postponed refresh commands. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 291, 18 pages.

[13] Finn de Ridder, Patrick Jattke, and Kaveh Razavi. 2025. Softhammer: Exploiting Rowhammer Bit Flips without Crashing. In *5th Workshop on DRAM Security (DRAMSec)*. ETH Zurich.

[14] Timothy J Dell. 1997. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics division* 11, 1-23 (1997), 5–7.

[15] Andrea Di Dio, Koen Koning, Herbert Bos, and Cristiano Giuffrida. 2023. Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks.. In *NDSS*.

[16] Jianshuo Dong, Han Qiu, Yiming Li, Tianwei Zhang, Yuanjie Li, Zeqi Lai, Chao Zhang, and Shu-Tao Xia. 2023. One-bit flip is all you need: When bit-flip attack meets model training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4688–4698.

[17] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2023. Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 45, 13 pages. doi:10.1145/3579371.3589080

[18] Ali Fakhrzadehgan, Yale N Patt, Prashant J Nair, and Moinuddin K Qureshi. 2022. Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 373–386.

[19] Carina Fiedler, Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Hannes Weissteiner, Abdullah Giray Yağlıkçı, Florian Adamsky, and Daniel Gruss. 2026. Memory Band-Aid: A Principled Rowhammer Defense-in-Depth. In *Network and Distributed System Security (NDSS) 2026*.

[20] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *2018 ieee symposium on security and privacy (sp)*. IEEE, 195–210.

[21] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*. 747–762. doi:10.1109/SP40000.2020.00090

[22] Google. [n. d.]. Share GPUs across workloads with GPU time-sharing. https://cloud.google.com/kubernetes-engine/docs/how-to/timesharing-gpus. Accessed: 2025-01-22.

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

[24] Zhongshu Gu, Enriquillo Valdez, Salman Ahmed, Julian James Stephen, Michael Le, Hani Jamjoom, Shixuan Zhao, and Zhiqiang Lin. 2025. NVIDIA GPU confidential computing demystified. *arXiv preprint arXiv:2507.02770* (2025).

[25] Yanan Guo, Zhenkai Zhang, and Jun Yang. 2024. GPU Memory Exploitation for Fun and Profit. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 4033–4050. https://www.usenix.org/conference/usenixsecurity24/presentation/guo-yanan

[26] Sudhanva Gurumurthi, Kijun Lee, Munseon Jang, Vilas Sridharan, Aaron Nygren, Yesin Ryu, Kyomin Sohn, Taekyun Kim, and Hoeju Chung. 2021. HBM3 RAS: Enhancing resilience at scale. *IEEE Computer Architecture Letters* 20, 2 (2021), 158–161.

[27] R. W. Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal* 29, 2 (1950), 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x

[28] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. 2021. Uncovering In-DRAM RowHammer Protection Mechanisms:A New Methodology, Custom RowHammer Patterns, and Implications. In *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Association for Computing Machinery, New York, NY, USA, 1198–1213. doi:10.1145/3466752.3480110

[29] Martin Heckel, Nima Sayadi, Jonas Juffinger, Carina Fiedler, Daniel Gruss, and Florian Adamsky. 2026. FLIPPYRAM: A Large-Scale Study of Rowhammer Prevalence. In *Network and Distributed System Security (NDSS) 2026*.

[30] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 497–514. https://www.usenix.org/conference/usenixsecurity19/presentation/hong

[31] Yichang Hu, Noah Brown, Yuhang Chen, Joshua Bakita, Tianlong Chen, Daniel Genkin, and Andrew Kwong. 2026. GDDRHammer: Greatly Disturbing DRAM Rows — Cross-Component Rowhammer Attacks from Modern GPUs. In *Proceedings of the 2026 IEEE Symposium on Security and Privacy (SP)*.

[32] Ilya-Zlobintsev. 2026. GitHub - ilya-zlobintsev/LACT: Linux GPU Configuration And Monitoring Tool. https://github.com/ilya-zlobintsev/LACT

[33] Aamer Jaleel, Gururaj Saileshwar, Stephen W. Keckler, and Moinuddin Qureshi. 2024. PrIDE: Achieving Secure Rowhammer Mitigation with Low-Cost In-DRAM Trackers. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 1157–1172. doi:10.1109/ISCA59077.2024.00087

[34] Patrick Jattke, Michele Marazzi, Flavien Solt, Max Wipfli, Stefan Gloor, and Kaveh Razavi. 2025. {McSee}: Evaluating Advanced Rowhammer Attacks and Defenses via Automated {DRAM} Traffic Analysis. In *34th USENIX Security Symposium (USENIX Security 25)*. 5621–5640.

[35] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. 2022. BLACKSMITH: Scalable Rowhammering in the Frequency Domain. In *2022 IEEE Symposium on Security and Privacy (SP)*. 716–734. doi:10.1109/SP46214.2022.9833772

[36] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. 2024. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 1615–1633. https://www.usenix.org/conference/usenixsecurity24/presentation/jattke

[37] JEDEC. 2023. GDDR6 Specification (JESD250D). (2023).

[38] JEDEC. 2023. HBM3 Specification (JESD238A). (2023).

[39] JEDEC. 2024. DDR5 Specification Update (JESD79-5C). (2024).

[40] JEDEC. 2026. GDDR7 Specification (JESD239D). (2026).

[41] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. 2023. CSI: Rowhammer–Cryptographic security and integrity against rowhammer. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1702–1718.

[42] Nureddin Kamadan, Walter Wang, Stephan van Schaik, Christina Garman, Daniel Genkin, and Yuval Yarom. 2025. ECC. fail: Mounting Rowhammer Attacks on DDR4 Servers with ECC Memory. In *34th USENIX Security Symposium (USENIX Security 25)*. 5679–5698.

[43] Jumin Kim, Seungmin Baek, Hwayong Nam, Minbok Wi, Nam Sung Kim, and Jung Ho Ahn. 2026. PVAC: A RowHammer Mitigation Architecture Exploiting Per-victim-row Counting. arXiv:2604.20576 [cs.CR] https://arxiv.org/abs/2604.20576

[44] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 638–651. doi:10.1109/ISCA45697.2020.00059

[45] M. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. Ham, J. W. Lee, and J. Ahn. 2022. Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging

Managed Refresh. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[46] Michael Jaemin Kim, Minbok Wi, Jaehyun Park, Seoyoung Ko, Jaeyoung Choi, Hwayoung Nam, Nam Sung Kim, Jung Ho Ahn, and Eojin Lee. 2023. How to kill the second bird with one ecc: The pursuit of row hammer resilient dram. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 986–1001.

[47] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*. 361–372.

[48] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. 2022. Half-Double: Hammering From the Next Row Over. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3807–3824. https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double

[49] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2018. ZebRAM: comprehensive and compatible software protection against rowhammer attacks. In *OSDI 18*.

[50] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. 2014. Stealing Webpages Rendered on Your Browser by Exploiting GPU Vulnerabilities. In *2014 IEEE Symposium on Security and Privacy*. 19–33. doi:10.1109/SP.2014.9

[51] Jie Li, Longda Zhou, Sheng Ye, Zheng Qiao, and Zhigang Ji. 2024. Understanding the Competitive Interaction in Leakage Mechanisms for Effective Row Hammer Mitigation in Sub-20 nm DRAM. *IEEE Electron Device Letters* 45, 1 (2024), 40–43. doi:10.1109/LED.2023.3334763

[52] Xiang Li, Ying Meng, Junming Chen, Lannan Luo, and Qiang Zeng. 2025. {Rowhammer-Based} Trojan Injection: One Bit Flip Is Sufficient for Backdooring {DNNs}. In *34th USENIX Security Symposium (USENIX Security 25)*. 6319–6337.

[53] Chris S Lin, Joyce Qu, and Gururaj Saileshwar. 2025. GPUHammer: Rowhammer Attacks on GPU Memories are Practical. In *34th USENIX Security Symposium (USENIX Security 25)*. 5719–5738.

[54] Chris S. Lin, Yuqin Yan, Guozhen Ding, Joyce Qu, Joseph Zhu, David Lie, and Gururaj Saileshwar. 2026. GPUBreach Code Repository. https://github.com/sith-lab/gpubreach. Accessed: 2026-04-27.

[55] Chris S. Lin, Yuqin Yan, Guozhen Ding, Joyce Qu, Joseph Zhu, David Lie, and Gururaj Saileshwar. 2026. GPUBreach: Privilege Escalation Attacks on GPUs using Rowhammer. In *Proceedings of the 2026 IEEE Symposium on Security and Privacy (SP)*.

[56] William Liu, Joseph Ravichandran, and Mengjia Yan. 2023. Entrybleed: A universal kaslr bypass against kpti on linux. In *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*. 10–18.

[57] Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci. 2023. Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer. In *29th Symposium on Operating Systems Principles (SOSP)*.

[58] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. 2023. Rowpress: Amplifying read disturbance in modern dram chips. In *50th International Symposium on Computer Architecture (ISCA)*.

[59] Haocong Luo, İsmail Emir Yüksel, Ataberk Olgun, A. Giray Yağlıkçı, and Onur Mutlu. 2025. Revisiting DRAM Read Disturbance: Identifying Inconsistencies Between Experimental Characterization and Device-Level Studies. In *2025 IEEE 43rd VLSI Test Symposium (VTS)*. 1–8. doi:10.1109/VTS65138.2025.11022861

[60] Evgeny Manzhosov and Simha Sethumadhavan. 2024. Polymorphic Error Correction. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 246–262.

[61] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. 2022. Protrr: Principled yet optimal in-dram target row refresh. In *IEEE Symposium on Security and Privacy (SP)*. 735–753.

[62] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. 2023. REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations. In *IEEE Symposium on Security and Privacy (SP)*.

[63] A Theodore Markettos, Colin Rothwell, Brett F Gutstein, Allison Pearce, Peter G Neumann, Simon W Moore, and Robert NM Watson. 2019. Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals. In *Proceedings of Network and Distributed Systems Security (NDSS) Symposium*.

[64] Maccoy Merrell, Daniel Puckett, Gino Chacon, Jeffrey Stuecheli, Stavros Kalafatis, and Paul V. Gratz. 2026. ORAP: Optimized Row Access Prefetching for Rowhammer-mitigated Memory. arXiv:2602.13434 [cs.AR] https://arxiv.org/abs/2602.13434

[65] Diego Meyer, Patrick Jattke, Michele Marazzi, Salman Qazi, Daniel Moghimi, and Kaveh Razavi. 2026. Phoenix: Rowhammer Attacks on DDR5 with Self-Correcting Synchronization. In *S&P (Oakland)*.

[66] Prashant J Nair, Vilas Sridharan, and Moinuddin K Qureshi. 2016. XED: Exposing on-die error detection information for strong memory reliability. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 341–353.

[67] Ravan Nazaraliyev, Yicheng Zhang, Sankha Baran Dutta, Andres Marquez, Kevin Barker, and Nael Abu-Ghazaleh. 2025. Not so refreshing: attacking GPUs using RFM rowhammer mitigation. In *Proceedings of the 34th USENIX Conference on Security Symposium (SEC)*.

[68] NVIDIA. 2019. Pascal MMU Format. https://github.com/NVIDIA/open-gpu-doc/blob/master/pascal/gp100-mmu-format.pdf.

[69] NVIDIA. 2025. NVIDIA GPU Memory Error Management. https://docs.nvidia.com/deploy/a100-gpu-mem-error-mgmt/overview.html Accessed: 2026-04-21.

[70] NVIDIA. 2025. Security Notice: Rowhammer – July 2025. https://nvidia.custhelp.com/app/answers/detail/a_id/5671.

[71] NVIDIA. 2026. CUDA C++ Best Practices Guide. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html Accessed: 2026-04-21.

[72] Ataberk Olgun, F. Nisa Bostanci, Ismail Emir Yuksel, Oguzhan Canpolat, Haocong Luo, Geraldo F. Oliveira, A. Giray Yaglikci, Minesh Patel, and Onur Mutlu. 2025. Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance. arXiv:2502.13075 [cs.AR] https://arxiv.org/abs/2502.13075

[73] Ataberk Olgun, Majd Osseiran, A Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. 2023. An experimental analysis of RowHammer in HBM2 DRAM chips. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 151–156.

[74] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, and Onur Mutlu. 2024. ABACuS: all-bank activation counters for scalable and low overhead RowHammer mitigation. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 89, 18 pages.

[75] Lois Orosa, Abdullah Giray Yaglikci, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S. Kim, and Onur Mutlu. 2021. A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21)*. ACM, 1182–1197. doi:10.1145/3466752.3480069

[76] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. 2020. Graphene: Strong yet Lightweight Row Hammer Protection. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[77] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 565–581. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[78] Moinuddin Qureshi. 2021. Rethinking ECC in the era of row-hammer. In *DRAMSec 2021*.

[79] Moinuddin Qureshi. 2025. AutoRFM: Scaling Low-Cost in-DRAM Trackers to Ultra-Low Rowhammer Thresholds . In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 991–1004. doi:10.1109/HPCA61900.2025.00078

[80] Moinuddin Qureshi and Salman Qazi. 2025. Moat: Securely mitigating rowhammer with per-row activation counters. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 698–714.

[81] Moinuddin Qureshi, Salman Qazi, and Aamer Jaleel. 2024. MINT: Securely Mitigating Rowhammer with a Minimalist in-DRAM Tracker . In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 899–914. doi:10.1109/MICRO61859.2024.00071

[82] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J. Nair. 2022. Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. doi:10.1145/3470496.3527421

[83] Moinuddin K. Qureshi. 2026. SALT: Track-and-Mitigate Subarrays, Not Rows, for Blast-Radius-Free Rowhammer Defense. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–16. doi:10.1109/HPCA68181.2026.11408602

[84] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1211–1220.

[85] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a Needle in the Software Stack. In *25th USENIX Security Symposium (USENIX Security)*. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[86] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J. Nair. 2022. Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows. In *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[87] Anish Saxena, Saurav Mathur, and Moinuddin Qureshi. 2024. 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* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1014–1028. doi:10.1145/3620665.3640404

[88] Anish Saxena, Gururaj Saileshwar, Jonas Juffinger, Andreas Kogler, Daniel Gruss, and Moinuddin Qureshi. 2023. PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 95–108. doi:10.1109/DSN58367.2023.00022

[89] Anish Saxena, Gururaj Saileshwar, Prashant J. Nair, and Moinuddin Qureshi. 2022. AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 108–123. doi:10.1109/MICRO56248.2022.00022

[90] Anish Saxena, Walter Wang, and Alexandros Daglis. 2025. Citadel: Rethinking Memory Allocation to Safeguard Against Inter-Domain Rowhammer Exploits. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1117–1131.

[91] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. Google Project Zero. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html Accessed: 2025-11-05.

[92] Mrityunjay Shukla, Shubham Roy, Sayandeep Saha, and Biswabandan Panda. 2026. PRowhammer: Propagating Bit-flips from CPU to GPU. (2026).

[93] Tyler Sorensen and Heidy Khlaaf. 2024. LeftoverLocals: Listening to LLM Responses Through Leaked GPU Local Memory. In *arXiv preprint arXiv:2401.16603*. https://arxiv.org/abs/2401.16603

[94] Michael B Sullivan, Mohamed Tarek Ibn Ziad, Aamer Jaleel, and Stephen W Keckler. 2023. Implicit memory tagging: No-overhead memory safety using alias-free tagged ecc. In *50th Annual International Symposium on Computer Architecture (ISCA)*.

[95] Hritvik Taneja, Ali Hajiabadi, Michele Marazzi, Kaveh Razavi, and Moinuddin Qureshi. 2026. MIRZA: Efficiently Mitigating Rowhammer with Randomization and ALERT. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–13. doi:10.1109/HPCA68181.2026.11408571

[96] Hritvik Taneja and Moin Qureshi. 2025. DREAM: Enabling Low-Overhead Rowhammer Mitigation via Directed Refresh Management. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 776–792. doi:10.1145/3695053.3731117

[97] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. In *2018 USENIX Annual Technical Conference (USENIX ATC)*. https://www.usenix.org/conference/atc18/presentation/tatar

[98] TechPowerUp. 2025. NVIDIA NVFlash. https://www.techpowerup.com/download/nvidia-nvflash/ Accessed: 2026-04-21.

[99] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. https://doi.org/10.1145/2976749.2978406

[100] Victor Van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. 2018. GuardION: Practical mitigation of DMA-based rowhammer attacks on ARM. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 92–113.

[101] Andrew J. Walker, Sungkwon Lee, and Dafna Beery. 2021. On DRAM Rowhammer and the Physics of Insecurity. *IEEE Transactions on Electron Devices* 68, 4 (2021), 1400–1410. doi:10.1109/TED.2021.3060362

[102] Junpeng Wan, Yanan Guo, Zhi Zhang, Zhuo Li, Dave (Jing) Tian, and Zhenkai Zhang. 2026. GeForge: Hammering GDDR Memory to Forge GPU Page Tables for Fun and Profit. In *Proceedings of the 2026 IEEE Symposium on Security and Privacy (SP)*.

[103] Alan Wang, Pranav Gopalkrishnan, Yingchen Wang, Christopher W. Fletcher, Hovav Shacham, David Kohlbrenner, and Riccardo Paccagnella. 2025. Pixnapping: Bringing Pixel Stealing out of the Stone Age. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[104] Yingchen Wang, Riccardo Paccagnella, Zhao Gang, Willy R. Vasquez, David Kohlbrenner, Hovav Shacham, and Christopher W. Fletcher. 2024. GPU.zip: On the Side-Channel Implications of Hardware-Based Graphical Data Compression. In *2024 IEEE Symposium on Security and Privacy (SP)*. 3716–3734. doi:10.1109/SP54263.2024.00084

[105] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing Deep-Learning Model Secret with GPU Context-Switching Side-Channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 125–137. doi:10.1109/DSN48063.2020.00031

[106] Minbok Wi, Jaehyun Park, Seoyoung Ko, Michael Jaemin Kim, Nam Sung Kim, Eojin Lee, and Jung Ho Ahn. 2023. SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In *HPCA*.

[107] Minbok Wi, Yoonyul Yoo, Yoojin Kim, Jaeho Shin, Jumin Kim, Yesin Ryu, Saeid Gorgin, Jung Ho Ahn, and Jungrae Kim. 2026. RowArmor: Efficient and Comprehensive Protection Against DRAM Disturbance Attacks. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 1640–1659. doi:10.1145/3779212.3790213

[108] Jeonghyun Woo, Junsu Kim, Aamer Jaleel, and Prashant Nair. 2026. Loaded Dice: Solving the Non-Selection Problem for Scalable Probabilistic RowHammer Defense. 1562–1578. doi:10.1109/ISCA66397.2026.00115

[109] Jeonghyun Woo, Shaopeng Chris Lin, Prashant J Nair, Aamer Jaleel, and Gururaj Saileshwar. 2025. Qprac: Towards secure and practical prac-based rowhammer mitigation using priority queues. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[110] Jeonghyun Woo and Prashant J. Nair. 2025. DAPPER: A Performance-Attack-Resilient Tracker for RowHammer Defense. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1005–1020. doi:10.1109/HPCA61900.2025.00079

[111] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. 2023. Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In *HPCA*.

[112] Steven C. Woo, Wendy Elsasser, Mike Hamburg, Eric Linstadt, Michael R. Miller, Taeksang Song, and James Tringali. 2024. RAMPART: RowHammer Mitigation and Repair for Server Memory Systems. In *Proceedings of the International Symposium on Memory Systems* (Alexandria, VA, USA) *(MEMSYS '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 15 pages. doi:10.1145/3631882.3631886

[113] Runjin Wu, Meng Zhang, You Zhou, Changsheng Xie, and Fei Wu. 2026. APT: Securing Against DRAM Read Disturbance via Adaptive Probabilistic In-DRAM Trackers. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 137–156. doi:10.1145/3779212.3790126

[114] Rui Xie, Asad Ul Haq, Yunhua Fang, Linsen Ma, Sanchari Sen, Swagath Venkataramani, Liu Liu, and Tong Zhang. 2025. Breaking the HBM Bit Cost Barrier: Domain-Specific ECC for AI Inference Infrastructure. arXiv:2507.02654 [cs.AR] https://arxiv.org/abs/2507.02654

[115] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In *USENIX Security*. https://www.usenix.org/conference/usenixsecurity20/presentation/yao

[116] A. Giray Yağlikçi, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. 2021. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *HPCA*.

[117] Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Geraldo F. Oliveira, İsmail Emir Yüksel, Ataberk Olgun, Haocong Luo, and Onur Mutlu. 2024. Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 560–577. doi:10.1109/HPCA57654.2024.00048

[118] Ismail Emir Yuksel, Ataberk Olgun, Nisa Bostanci, Haocong Luo, Abdullah Giray Yaglikci, and Onur Mutlu. 2025. ColumnDisturb: Understanding Column-based Read Disturbance in Real DRAM Chips and Implications for Future Systems. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 975–994.

[119] Yicheng Zhang, Ravan Nazaraliyev, Sankha Baran Dutta, Andres Marquez, Kevin Barker, and Nael Abu-Ghazaleh. 2025. NVBleed: Covert and Side-Channel Attacks on NVIDIA Multi-GPU Interconnect. *arXiv preprint arXiv:2503.17847* (2025).

[120] Zhenkai Zhang, Tyler Allen, Fan Yao, Xing Gao, and Rong Ge. 2023. TunneLs for Bootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. 15 pages. doi:10.1145/3576915.3616672

[121] Zhao Zhang and Nikolai Iliev. 2026. Using Canary Cell to Detect and Prevent Rowhammer Attack. *IEEE Micro* (2026), 1–11. doi:10.1109/MM.2026.3681120

[122] Longda Zhou, Jie Li, Zheng Qiao, Pengpeng Ren, Zixuan Sun, Jianping Wang, Blacksmith Wu, Zhigang Ji, Runsheng Wang, Kanyu Cao, and Ru Huang. 2023.

Double-sided Row Hammer Effect in Sub-20 nm DRAM: Physical Mechanism, Key Features and Mitigation. In *2023 IEEE International Reliability Physics Symposium (IRPS)*. 1–10. doi:10.1109/IRPS48203.2023.10117677

[123] Zhe Zhou, Wenrui Diao, Xiangyu Liu, Zhou Li, Kehuan Zhang, and Rui Liu. 2016. Vulnerable GPU Memory Management: Towards Recovering Raw Data from GPU. In *arXiv preprint arXiv:1605.06610*. https://arxiv.org/abs/1605.06610

A Open Science

Our open-sourced artifact contains the code for our main experimental results, namely the code for the Rowhammer attack and campaigns with GPUThor (used to generate Table 3), the reverse engineering results (Table 2 and Figures 4, 6, 7), Rowhammer campaign with ECC-enabled (Section 7&8), and the privilege escalation exploit with ECC-enabled (Section 8) that builds on code from prior work [54]. We plan to open-source our artifacts after the completion of the coordinated disclosure process and artifact evaluation.

B Ethical Considerations

Responsible disclosure. We responsibly disclosed our findings to NVIDIA and the major cloud providers (Google, Microsoft, AWS) prior to the public disclosure, and aligned on a coordinated disclosure plan. We have also reached out to the cloud GPU provider whose GPUs had ECC disabled by default.

Minimizing real-world harm. All experiments were conducted on locally owned GPUs or on cloud GPUs within isolated VMs, without any impact to any other users. On the cloud GPUs, where our VM is provisioned with a root user by the cloud provider, we only attempted privilege escalation from a user process to root within the provisioned VM (obtaining privileges we were already given by the cloud vendor). We did not attempt any further elevation to hypervisor privilege or VM escape, to prevent harm to production systems. All ECC-enabled experiments (DoS and privilege escalation) were performed only on locally owned GPUs. No production systems or users were impacted through exploits or crashes caused by ECC errors (DUEs or SDCs).

C Generative AI Usage

We used GPT-5.2 and Claude Sonnet 4.6 and Opus 5 for minor grammatical corrections and light stylistic refinements of the manuscript. All such edits were carefully reviewed and verified by the authors.

D Evaluation on Other GPUs

We tested several other GPUs available through cloud providers. For each GPU in Table 6, we generate a Row Set for 1GB of memory and perform GPUThor across all intensity levels (2× to 6.6×). However, other than the A4000-A6000 with GDDR6 memory, none have bit flips with our attack patterns, suggesting different TRR implementations that may require more sophisticated hammering or lack of a vulnerability to Rowhammer.

Table 6: Evaluations on Other GPUs

	GDDR6					GDDR6X	HBM2e
	3080 Mobile (GA104)	A10 (GA102)	A2000-6000 ADA (AD102-107)	L4 (AD102)	L40 (AD104)	4090 (AD102)	A30 (GA100)
Flips	No	No	No	No	No	No	No

E Data Pattern Dependency

Figure 12 shows the effect of aggressor and victim data patterns on the number of bit flips in GPUThor campaigns, previously discussed in Table 3. Consistent with prior work [31, 53], we see that victim/attacker data patterns of 0xff/00 induces the most bit flips, triggering more than half the flips observed in our campaigns.

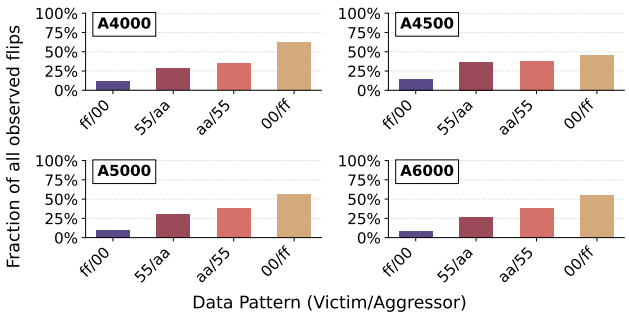

Figure 12: Percentage of Unique Flips Found per Data Pattern for each GPU.

We revisit the aggressor–victim distance distribution reported in prior work [31, 53], measured separately for each data pattern. Figure 13 shows that each data pattern yields a distinct distance distribution for bit flips. Most victim/aggressor data patterns trigger bit flips in victim rows that are logically at most 15 rows away from aggressor rows, validating that there is a non-linear mapping of logical rows to physical rows in the GDDR6 DRAM, as previously observed by GDDRHammer [31].

However, the victim/aggressor data pattern 0xAA/55 has bit flips only when an aggressor row is within a logical distance of 3. This suggests that certain physical rows (where the logical distances between aggressor and victim are larger than 3) are not vulnerable to this data pattern, as they are to other data patterns. We hypothesize that this could be due to differences in data scramblers used in different parts of the bank. This data dependence in the Rowhammer susceptibility, linked to spatial locations in the bank, could lead to new side-channels that leak the data in neighboring rows. We leave explorations of this for future work.

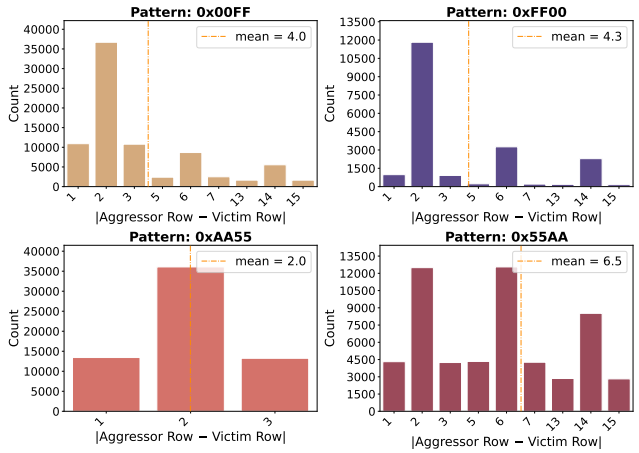

Figure 13: Aggressor-to-Victim Distance distribution for different Data Patterns. Data collected from bit flips occurring across all campaigns and GPUs.

F Unique Decoys Sensitivity Analysis

In Section 5, our attack patterns utilize unique, randomly selected decoy rows (one activation per decoy). However, for counter-based TRR designs [21], issuing more than one ACT per decoy row may evict aggressors more reliably from the TRR sampler. To evaluate this, we study whether the frequency of decoy activations influences attack effectiveness. For a given aggressor activation count in a 18 tREFI hammering pattern, we progressively reduce the number of unique decoy rows in the pattern (keeping the total decoy activations constant) until bit flips are no longer observed. Contrary to our hypothesis, Figure 14 shows that at higher aggressor activation counts, successful attacks require more unique decoys rather than repeated accesses to the same rows. Moreover, this threshold grows exponentially with increasing hammering intensity. Thus, unique decoys are preferred to fool the TRR in GDDR6 compared to repeated activations to the same decoy. This suggests that the TRR tracker in Samsung GDDR6 memories is likely counter-less or has a very small saturating counter per entry.

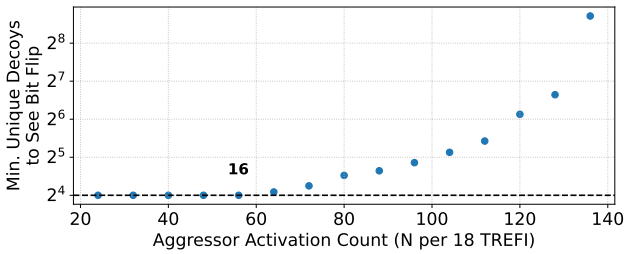

Figure 14: Minimum unique decoy rows required to trigger bit flip for the 18-TREFI pattern when we extend our hammering to higher intensity. At 56 ACTs or less, only 16 unique decoys (the tracker size) are needed, but when the aggressor activation counts are increased to 135 ACTs, up to 400 unique decoy rows are required to trigger bit flips.