PAGE404: Breaking GPU Isolation Through Hardware-assisted Page Migration

Ravan Nazaraliyev*, Yicheng Zhang†, and Nael Abu-Ghazaleh*
*Department of Computer Science and Engineering, University of California, Riverside
Email: rnaza005@ucr.edu, naelag@ucr.edu
†Department of Electrical and Computer Engineering, George Mason University
Email: yzhang95@gmu.edu

***Abstract*—NVIDIA's Grace Hopper (GH200) superchip represents a new generation of heterogeneous computing, tightly integrating a 72-core Arm-based CPU with a Hopper GPU on a single module via a high-bandwidth, cache-coherent interconnect. By eliminating the conventional PCIe bottleneck that has long constrained CPU–GPU data movement, this design establishes a unified memory domain that scales to the capacities demanded by generative AI and high-performance computing (HPC) workloads. The architecture also introduces hardware-assisted page migration support: driven by access counters, the system dynamically promotes frequently accessed pages from host CPU memory to the GPU DRAM to improve memory locality. In this paper, we study the security of this emerging architecture against new types of side- and covert-channel attacks targeting its page migration support. We analyze counter-driven migration across both the hardware mechanisms and the UVM driver's interrupt-handling pipeline, and identify two side-channel vulnerabilities: one rooted in the shared structure of access counters, and the other in the driver's response to migration interrupts. We further show that both channels are exploitable across Multi-Instance GPU (MIG) partitions, a feature intended to provide strong, hardware-enforced isolation between co-located tenants. Using proof-of-concept attacks, we build covert channels that reach roughly 8,500 bits per second through eviction-based signaling and 8,100 bits per second through migration-delay-based signaling across MIG instances. The interrupt-based channel also fingerprints inference workloads across distinct large language models (LLMs) and CPU SPEC2017 benchmarks, and estimates the prompt length processed during LLM inference with high accuracy. Taken together, these results expose a security gap in hardware-managed heterogeneous memory systems and call into question the isolation guarantees of current MIG-enabled multi-tenant GPU deployments. The artifact of PAGE404 is available at PAGE404 GitHub.**

1. Introduction

The growing demands of machine learning models and high-performance computing (HPC) workloads challenge the capabilities of modern CPUs, GPUs, and their memory systems. Successive GPU generations have been developed with increasing computational and memory capabilities in an effort to meet these needs. A significant source of overhead arises from memory management across the GPU and CPU: existing memory management approaches such as *Unified Virtual Memory* (UVM) require expensive software-managed fault handling via the UVM driver to transfer pages from/to the CPU and GPU memories. With the large memory demands of large models (e.g., to hold the KV cache), efficient integration between the GPU and CPU is needed to allow the GPU to use the host CPU memory.

To address these demands, newer designs, such as NVIDIA's Grace Hopper (GH200) superchip, support tighter CPU–GPU integration and hardware-managed memory systems to enable low-overhead uniform memory access between the GPU and CPU. The GH200 integrates an ARM-based Grace CPU with a Hopper GPU via NVLink-C2C, a cache-coherent, high-bandwidth interconnect supporting the Address Translation Service (ATS) [1]. ATS enables GPU threads to access system-allocated memory directly through hardware-assisted virtual-to-physical address translation, eliminating most of the software managed page-fault overhead of conventional UVM. The GH200 further introduces a hardware-assisted page migration policy that tracks page usage and automatically migrates hot pages to the GPU. This policy is implemented through *access counters*: per-region hardware counters that track GPU accesses to system memory pages and notify the UVM driver once a configurable threshold is reached. The driver then migrates the corresponding pages from host DRAM to GPU-local memory, optimizing memory locality without requiring programmer intervention. While this mechanism is designed for performance, shared access counters and the driver's interrupt-handling pipeline for servicing these notifications are subject to contention from co-located GPU workloads, causing observable migration delays related to other system activity.

In this work, we investigate whether these new mechanisms for hardware-assisted page-migration expose side channel and covert channel vulnerabilities. Because these platforms are expensive, they are frequently shared among potentially mistrusting users (e.g., in a cloud deployment). We reverse-engineer the access-counter subsystem and the UVM driver's interrupt-handling mechanism, characterizing the structure and behavior of the page migration counters and how migration notifications are processed by the driver.

From this analysis, we identify two previously unexplored side-channel vulnerabilities.

The first channel stems from the shared structure of the access-counter hardware across tenants. We discover that the counter hardware is organized into a set-associative structure similar to caches, and notifications are triggered deterministically once a migration threshold is reached. Since this structure is shared, it can be exploited using contention (e.g., evicting other processes' counters) for covert communication across tenants. Similarly, the leakage through these shared counters can be measured (observed through the presence or absence of migration) to identify victim accesses to the same counter set. The second channel exploits migration contention on the shared driver: calls to the driver (e.g., to launch a new kernel) can be detected since they affect the response latency to migration events. Crucially, both channels target a GPU-wide shared resource, which makes them exploitable *across Multi-Instance GPU (MIG) partitions*, breaking the isolation guarantees of MIG. On GH200, MIG partitions are shared across two containers, which is currently the only mechanism by which MIG instances are exposed to users.

Building on these findings, we demonstrate several covert- and side-channel attacks. The eviction-based covert channel achieves 8500 bps at a 1.5% error rate, rising to 15500 bps at roughly 5% error with two eviction sets, while the eviction-based side channel distinguishes among 10 different CUDA workloads with 94% accuracy when they use system-allocated memory. The migration-delay leakage enables a covert channel of 8100 bps at a 4% error rate and, across several side-channel attacks, distinguishes CUDA and LLM workloads with around or above 90% accuracy, estimating an LLM's processed token length with a mean absolute error of 4.6 tokens on average. Moreover, we show that even when a tenant runs no GPU workload at all, a spy can still infer side-channel information about that tenant's CPU workloads. This is because page migration requires the writer-prioritized mmap_lock to read the process's memory map. As a result, concurrent CPU memory operations (for example, munmap calls) can delay the migration's read request.

PAGE404 makes the following contributions:

- We reverse-engineer the complex hardware-assisted page migration support present on the new NVIDIA GH200 superchip, characterizing its underlying operation and discovering two new leakage vectors.
- We design covert channel attacks that enable information exchange across isolated tenants operating on separate Multi-Instance GPU (MIG) partitions.
- We demonstrate a cross-tenant side-channel attack between processes located on different MIG partitions that exploits two distinct leakage vectors to spy on different workloads. Specifically, we accurately fingerprint a large language model (LLM) architecture and infer some information about the prompt executed by a victim workload.

Responsible Disclosure. We disclosed our findings to NVIDIA. In their response, NVIDIA acknowledged our findings and indicated that they do not consider this a security issue, stating that the observed side-channel leakage is coarse-grained and noting that NVLink is not shared across untrusted tenants [1].

2. Background and Threat Model

This section overviews the NVIDIA memory management and outlines the threat model and attack assumptions.

2.1. Background

Memory management across heterogeneous processors such as CPUs and GPUs provides a unified abstraction over what are physically distributed memory resources (separate GPU and CPU memories), allowing automated management of memory without expensive run-times or burdensome programmer management. NVIDIA's Unified Virtual Memory (UVM) [2] was an early step in this direction, allowing CPU and GPU to share a single virtual address space. Heterogeneous Memory Management (HMM) [3], a later model, enables GPU threads to access system-allocated memory directly via shared virtual addressing. The overheads of memory management continue to be high, leading to more recent architectures, such as the Grace Hopper GH200 superchip, that incorporate hardware-assisted address translation and dynamic page migration [1], [4] to improve performance and to provide coherence across an address space that spans the CPU and GPU memories.

Workloads whose memory footprint exceeds GPU capacity can benefit from this tight integration. Consider the key-value (KV) cache in large language model inference: as sequence lengths and batch sizes grow, the KV cache can quickly outgrow the available GPU memory [5], forcing developers to either cap context length or manually stage data between host and device. Large-scale graph processing poses a similar challenge, where the graph's vertex and edge data may be far larger than GPU memory [6], [7], yet access patterns are irregular and data-dependent, making static partitioning ineffective. In both cases, the programmer is traditionally burdened with explicit memory management, allocating device buffers, orchestrating host-to-device transfers, and tracking which data resides where. This motivates the critical primitives these systems aim to provide: a unified address space so CPU and GPU can reference the same data through a single pointer, on-demand paging so data migrates automatically only when accessed, and transparent oversubscription so the working set can exceed physical GPU memory without manual intervention. The systems described below address these needs to varying degrees.

UVM Demand-Fault Migration. Figure 1 illustrates a high-level overview of address translation and migration

1. We note that this rationale overlooks key aspects of our threat model: PAGE404 does not require physical NVLink interconnect sharing between victim and attacker tenants. Furthermore, in multi-GH200 configurations, cross-tenant driver-level interaction and concurrent memory management operations can induce contention and observable migration side effects even without shared physical interconnects.

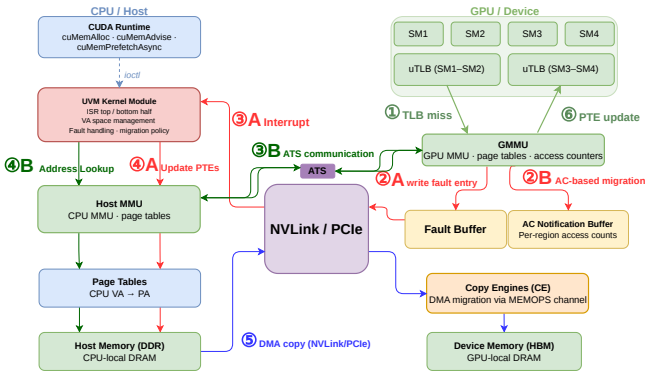


Figure 1: Unified Virtual Memory diagram.

paths present in UVM implementations. On the CPU side, the CUDA Runtime exposes a programmer-facing API for memory allocation and it communicates with the Kernel Module via `ioctl` system calls, which manage the virtual-address (VA) space mapping and migration policies. On the GPU side, Streaming Multiprocessors (SMs) each support their address translations through a micro-TLB (uTLB), which are managed by the GPU Memory Management Unit (GMMU) responsible for maintaining the GPU page tables. When a GPU thread accesses a page not currently mapped in its uTLB, a TLB miss is raised ① and forwarded to the GMMU. The GMMU writes a structured fault entry into the Fault Buffer: a GPU-visible ring buffer shared with the CPU ②A. Subsequently, it raises an interrupt to the host ③A. The UVM Kernel Module services this interrupt in its bottom half, draining the Fault Buffer. The module then updates the CPU-side page tables to reflect the new page placement ④A, and instructs the Copy Engines (CE) to perform a direct memory transfer of the faulting page over NVLink or PCIe from Host Memory to Device Memory (HBM) ⑤. Once the copy completes, the GMMU updates its own page table entries and invalidates the relevant uTLB entries so that the faulting instruction can be replayed transparently ⑥.

Heterogeneous Memory Management (HMM). HMM extends UVM's shared virtual addressing to any system-allocated memory (ordinary `malloc`, `mmap`, etc.) by integrating with the OS page-fault subsystem, whereas UVM only works for memory explicitly allocated through `cudaMallocManaged`. While this may improve the programmer productivity, it leads to substantial overheads, which motivated the hardware-assisted memory management support present in the Grace Hopper superchip.

Hardware-assisted memory sharing. GPU memory accesses may trigger page faults that require the UVM/HMM kernel module to intervene, introducing interrupt-handling overheads and additional OS interactions that can substantially impact performance. UVM/HMM also lacks hardware cache coherence between the CPU and GPU, requiring developers to manage memory coherence manually or limit updates from both the CPU and the GPU. Given that optimal page placement is often unpredictable and may change at

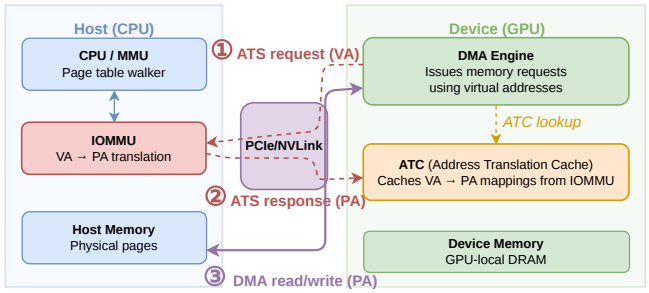


Figure 2: Address Translation Service in PCIe interconnect.

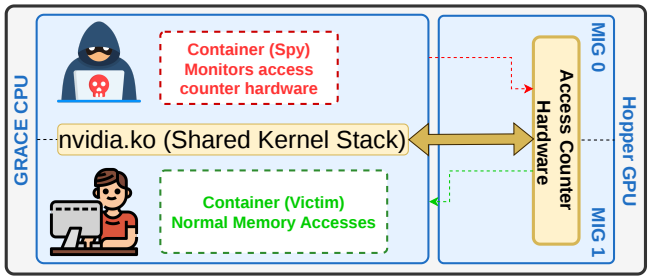


Figure 3: Threat model.

run time as access patterns change, static page placement is difficult.

As shown in Figure 2, the host side comprises the CPU and its Memory Management Unit (MMU), which owns the page tables, and an Input/Output MMU (IOMMU) that enforces address translation for device-initiated memory accesses. On the device side, the DMA Engine issues memory requests using virtual addresses and is backed by an on-chip Address Translation Cache (ATC) that stores recently resolved VA→PA mappings. When the DMA Engine needs to access a host memory location, it first checks its local ATC. On a miss, it sends an *ATS translation request* [8], [9] carrying the virtual address across the PCIe link to the host Root Complex, which forwards it to the IOMMU ①. The IOMMU walks the page tables, resolves the physical address, and returns an *ATS translation response* ②; the device caches this VA→PA mapping in its ATC for subsequent accesses. With a valid translation cached, the DMA Engine proceeds to issue physical-address DMA read/write operations directly to host memory over the PCIe link ③, bypassing the need for further translation requests. On the GH200, NVLink-C2C [10] extends this mechanism with cache coherence, enabling the GPU to access host memory without explicit page migration. Within UVM, this serves as the hardware-assisted alternative to the fault-and-copy path in Figure 1 (③B–⑥).

Access-Counter-Driven Migration. In parallel, the GMMU maintains hardware *access counters* that track how frequently the memory region is accessed remotely from the GPU. Once a configurable access threshold (default 256) is exceeded, the GMMU writes a notification entry into the AC Notification Buffer and asserts an interrupt ("counter ≥ threshold") ②B.

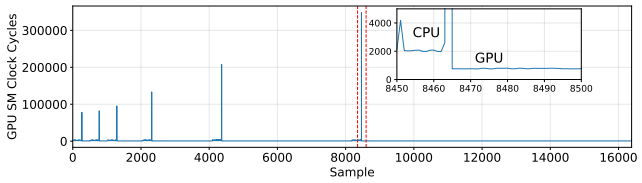


Figure 4: Hardware-assisted page migration on GH200.

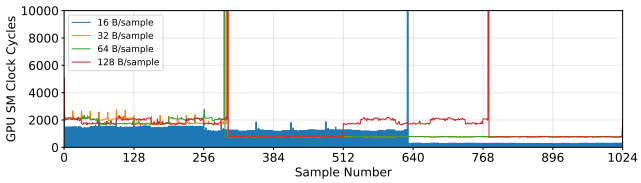


Figure 5: Varying size of the stride.

The UVM Kernel Module receives this as an *AC notification* and, rather than waiting for a hard fault, proactively issues a *GPFIFO push* command to the Copy Engines, triggering a migration of the frequently accessed pages to Device Memory ⑤. Unlike the demand-fault path, this migration does not stall the executing GPU thread; it is treated as a placement hint rather than a mandatory demand. The access counter structure is shared among MIG instances, and migration is handled in the driver kernel, shared among tenants, together with primary sources of contention analyzed in this work.

2.2. Threat Model

PAGE404 assumes the highest level of isolation between tenants that use different Multi-Instance GPU slices. MIGs are allocated to users in containers [11], and Kubernetes Engines can be utilized to handle the sharing mechanism [12]. NVIDIA also provides vGPU technology [13], where each user has its own guest driver, but it is not supported on ARM platforms. Hence, the highest isolation level is that each user gets its own MIG slice within their container. Containers share the same GPU driver (nvidia.ko in Fig. 3). Previous works assumed sharing of a GPU with other tenants [14], [15], [16], [17], [18], where a *spy* CUDA application and a *victim* GPU context, i.e., a CUDA or graphics application, share a GPU. Other works assumed GPU sharing in cloud settings, where multiple tenants can share the same physical GPU through virtualization [19], [20], [21], [22]. Additionally, prior research has demonstrated practical techniques for achieving co-location in cloud systems [23], [24], [25]. MIG attack papers [18], [26] assumed that different users get their own MIG instance. The attacker does not require superuser privileges. Instead, they perform experiments to reverse-engineer the page migration details and the behaviour of hardware access counters, as a preliminary step for the attacks.

3. Demystifying HW-assisted Page Migration

This section describes the methodology we use to reverse-engineer undocumented aspects of page migration and the hardware access-counter mechanism on the GH200 Superchip. Experiments are performed on GH200 nodes from both CloudLab [27] and Lambda Cloud. The modified driver used throughout this section is available in our artifact repository.

3.1. Page Migration Access Counters

Overview of the page migration. To understand the behavior of page migration, we design an experiment that performs strided memory accesses over a system-allocated (i.e., CPU allocated) memory region using a single GPU thread. The resulting access latencies reveal a repeatable migration pattern that recurs over every 2 MB region. Figure 4 depicts a representative 2 MB virtual address space, exposing the underlying migration mechanism. The migration behavior is explained by two complementary mechanisms: page migration driven by access counters, and prefetching performed by the UVM driver module. Prefetching is governed by a tree-based promotion policy implemented within the UVM driver. Migration operates at the granularity of a single 64 KB page: once the access counter for a given 64 KB page reaches a specific threshold, that page is migrated to GPU memory. With a sequential access pattern, upon each migration, potentially additional regions are prefetched corresponding to one node higher in the promotion tree, so that the number of prefetched 64 KB pages doubles at every migration event, following the sequence 1–1–2–4–8–16 pages until the full 2 MB region has been promoted to HBM. Under a random access page migration pattern, a subtree is prefetched if more than half its leaf 64KB pages have already been migrated.

Observation 1

Hardware-assisted page migration operates at a 64 KB page granularity. The prefetching scheme is the traditional tree-based UVM prefetcher, and prefetching occurs within a single 2 MB virtual page.

Determining the cacheline size. In this experiment, we vary the stride size to determine the cache line size of GPU memory accesses. Figure 5 shows access latencies for four different stride sizes. For stride sizes of 128, 64, and 32 bytes, the number of accesses required to trigger page migration remains constant at 256. However, when the stride size is reduced to 16 bytes, the number of accesses required to trigger page migration doubles. Furthermore, at a stride size of 16 bytes, the access latency alternates between that of CPU main memory (DDR) and GPU L1 cache, indicating that two consecutive 16-byte accesses fall within the same 32-byte cache line—with the first access fetching the line from DDR and the second being served from the L1 cache. This observation is consistent with prior work [16], [28], [29] and confirms that the GPU cache line

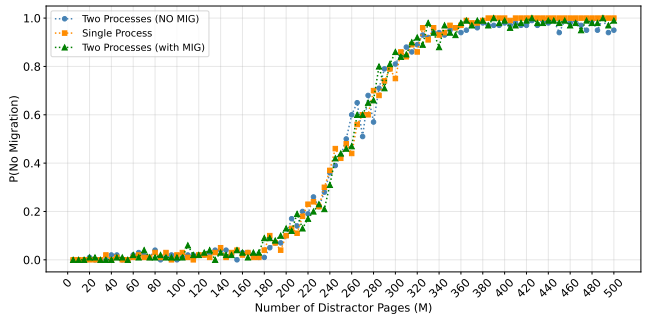


Figure 6: Probability of no migration of first page when M distractor pages are accessed in Phase 2.

size for system memory accesses is 32 bytes [30]. Finally, we repeat the experiment using `load` instructions annotated with the `volatile` keyword and observe that data is not cached at any memory hierarchy level when the GPU accesses system memory through `volatile` loads, bypassing the cache entirely.

Observation 2

The cacheline size is 32 bytes. The load operation used to access a *volatile* memory does not cache the data at any GPU cache level.

Hardware Access Counter Circuit and Sharing Mechanism. As described in [4], [31], hardware counters in GH200 are implemented as a finite pool, where each counter is assigned to a specific address range, either virtual or physical, according to a replacement policy such as Least Recently Used (LRU) [31]. In this experiment, we probe the size of this counter pool. The experiment proceeds as follows: a process first accesses $N - \delta$ addresses to bring a target page's counter close to the migration threshold. Using the default threshold of $N = 256$ and setting $\delta = 2$, only two additional accesses are needed to trigger hardware-assisted page migration for the target page. We then introduce M *distractor* pages, each accessed $X < N$ times, forcing the pool to assign counters to these new pages. After making X accesses to each of the M distractor pages, we perform the remaining δ accesses to the original target page and observe whether migration is triggered [2]. Figure 6 shows (Single process) the probability of hardware-assisted page migration as a function of M, the number of distractor pages. The results indicate that the counter pool is exhausted at approximately 360–380 distractor pages. The lack of a sharp cutoff in the figure is attributable to a behavior we characterize and explain later: distinct 64KB pages residing within the same 2MB page are assigned the same counter, producing an apparent eviction effect that is, in reality, a consequence of using the same counter.

Hardware Access Counter Sharing across 2 Processes and MIG instances. We extend the previous experiment to examine counter pool behavior across two independent processes, comparing execution on a full GH200 (without MIG partitioning) against execution on isolated Multi-Instance GPU (MIG) instances. The experimental setup is restructured into three phases: *Phase 1*, in which Process 1 probes its target page with $N - \delta$ accesses and then blocks awaiting a signal; *Phase 2*, in which Process 2 accesses its own M distractor pages; and *Phase 3*, in which Process 1 resumes and attempts to trigger migration on the target page. As shown in Figure 6, the observed behavior is consistent across both the non-MIG and MIG configurations, with migration probability degrading at the same range of $M \approx 360$–380 distractor pages regardless of whether the two processes are isolated within separate MIG instances. This demonstrates that the hardware counter pool is shared across MIG boundaries, meaning that MIG's process isolation guarantees do not extend to the page migration hardware counters.

Observation 3

The hardware counter pool is finite and is shared across Multi-Instance GPU (MIG) instances, violating the isolation guarantees expected of MIG partitioning.

Deep dive into Hardware Counters. Virtual or Physical addresses? While we have shown that hardware counters are shared across processes and MIG instances, this does not reveal how counters are assigned. Although Figure 6 shows that the counter pool saturates at approximately 380 entries, migrations still occasionally fail even beyond 400 distractor pages, suggesting a probabilistic assignment mechanism. This probabilistic behavior arises because counters may be assigned based on either virtual addresses (VA) or physical addresses (PA) [31], both of which are randomized across process invocations due to ASLR and OS memory allocation. To determine whether counter assignment is VA-based, we disable `ASLR`, ensuring that the process receives the same VA on every run. The experimental setup follows the previous experiment (single-process configuration) with this modification. If counters are assigned based on VA and the pool employs an LRU eviction policy, repeatedly running the process with fixed VAs should produce a deterministic eviction set of size E beyond which migration consistently fails. However, we observe entirely probabilistic behavior even with `ASLR` disabled, indicating that VA plays no role in hardware counter assignment.

Observation 4

Hardware access counters assignments are based on the physical addresses of pages.

Understanding Hardware Structure. Since the counter pool is finite, understanding the counter assignment policy reduces to finding an *eviction set*: the minimum number of distractor pages required to evict the counter assigned to a target page. This problem is analogous to cache eviction set construction, but introduces unique challenges: (1) once a page's counter reaches the migration threshold, the page is

2. Migration can be detected either by inspecting the page's physical address or by repeatedly accessing the page and determining whether it migrates within an acceptable number of memory accesses, a quantity we later denote as the migration delay.

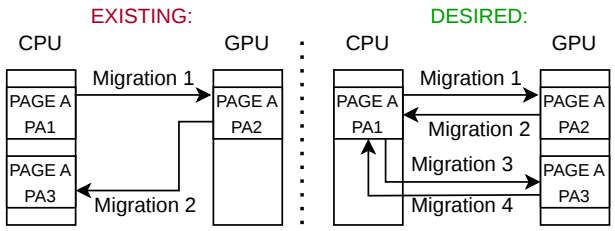


Figure 7: Memory migration mechanism and desired state.

migrated to GPU DRAM and is no longer resident in host memory; and (2) migrating a page back to host memory does not preserve its physical address, as the original physical frame is released to the OS and a new allocation is made upon return. One way to handle this is to pin the host page; however, pinning disables the page migration.

The CUDA API `cudaMemPrefetchAsync()` can be used to prefetch the page back to host memory[3]. However, as illustrated in Figure 7, once a page is migrated to the device, its host-side physical frame is freed and returned to the OS. Prefetching the page back to host memory (Migration 2 in the existing flow) triggers a fresh allocation, assigning an arbitrary new physical frame. To overcome this limitation, we instrument the open-source NVIDIA driver to implement a modified page migration scheme: when a host-allocated page is migrated to the GPU, whether via hardware counters or any other mechanism, its physical frame is not released to the OS; instead, it is retained in a *page cache*. Upon prefetching or migrating the page back to host memory, the cached physical frame is restored, ensuring physical address stability across migration events.These modifications enable eviction set construction experiments analogous to those used in cache side-channel research.

We construct the eviction set using the following procedure: the target page is first accessed $N - \delta$ times, bringing its counter close to the migration threshold; then M distractor pages are accessed; finally, the target page is accessed again to determine whether migration is triggered, following the methodology of previous experiments. The driver modification is employed to reset counters and bring pages back to the host between trials: each page is accessed N times to force its migration to GPU DRAM, after which it is prefetched back to host memory, restoring its physical frame and resetting its counter. We then determine the minimum number of distractor pages sufficient to consistently prevent the target page from migrating, which corresponds to the eviction set size. Additionally, by identifying the number of unique eviction sets, we derive the total number of hardware access counters. The results reveal that each counter set consists of 32 access counters (as illustrated in Fig. 8), with 8 unique sets, yielding a total of 256 hardware access counters across the GPU.

3. `cudaCpuDeviceId` is passed as the destination device identifier.

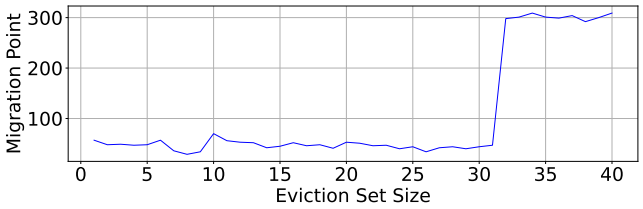


Figure 8: Hardware access counter set size. For illustration, we access the page continuously and observe when migration occurs; a value higher than 256 indicates that the counter had previously been evicted, so a fresh set of 256 accesses was required to trigger migration.

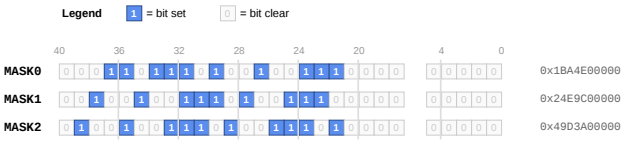


Figure 9: Bit-level view of each XOR mask (bit 39 = MSB $\rightarrow$ bit 0 = LSB).

Observation 5

The hardware access counters are distributed across 8 sets, with 32 counters in each set.

Understanding mapping of pages to counters. Prior work has reverse-engineered the hash functions that map PAs to DRAM banks [32], [33], or to LLC cache sets [34], [35] under the assumption that these functions are linear (i.e., XOR-based). Adopting the same assumption, we search candidate XOR functions against the addresses we previously identified as belonging to the same set. We use the functions to validate that our conflict sets are consistent. We then compute the null space of the resulting system, and as a final validation step, we verify that the recovered functions produce distinct set IDs across the 8 groups of PAs. Using the modified driver, we collect 8 groups comprising approximately 4,700 addresses in total and run the brute-force search. The procedure recovers exactly 3 independent functions in the null space, matching the expected number of selector bits ($8 = 2^3$). The recovered masks are shown in Figure 9. All three functions operate exclusively on bits 21 through 38, which implies that every 2 MB physically contiguous range is mapped to a single counter.

Counter assignment vs. page migration. Because counters are assigned at a 2 MB granularity, a discrepancy arises with the system page size, which also defines the default migration granularity of 64 KB. We design an experiment to determine how access to different 64 KB sub-regions within the same 2 MB range contributes to migration decisions. Across multiple trials, we observe that accesses to *any* 64 KB sub-region within a 2 MB range increment the same counter, and that migration is performed on the specific 64 KB region targeted by the most recent access.

Observation 6

Each access counter is owned by a 2 MB physical range, while migration is performed at the granularity of the most recently accessed 64 KB region within that range.

Effect of Counter Values on the Eviction Policy. Throughout our experimentation, we assumed a Least Recently Used (LRU) eviction policy and held the counter values constant across all page accesses, ensuring that our measurements were not confounded by variations in the counter state. In this experiment, we relax that assumption to determine whether the counter values themselves influence the eviction decision. We first construct an eviction set of 40 addresses, each residing in a distinct 2 MB region but mapping to the same counter set, following the methodology established in our earlier findings. We then execute a pair of nested loops: for a fixed number of accesses n to the target page, we sweep the number of accesses m to each page in the eviction set. To obtain noise-free measurements, we employ a second, independent eviction set to reset the counter values in the counter set between trials. Each trial first issues n accesses to the target page and subsequently issues m accesses to the eviction pages. Figure 10 reports the resulting behavior, where each line corresponds to a distinct counter value of the target (first) page. As we jointly vary n and m, we observe that the counter values do in fact govern the eviction decision. When a counter must be evicted, the victim is the counter holding the smallest value, provided that value is below 7. Once all counter values reach or exceed 7, the counter value ceases to influence the decision and eviction reverts to LRU. The transition points visible in Figure 10, where each curve drops from an eviction set size of 41 back to 32, shift rightward as the target page's counter value increases, confirming that a higher counter value confers stronger residency until the saturation threshold of 7 is reached. In summary, the access counters prioritize eviction by counter value when that value is below 7, and otherwise fall back to LRU.

Observation 7

The eviction policy evicts the counter with the smallest value when that value is below 7, and reverts to LRU otherwise.

Counter Persistence Across Process Launches. Having characterized the eviction policy within a single execution, we investigate whether hardware access counter state persists across process boundaries. Specifically, once a counter tracks a physical address (PA) during an initial execution, does that accumulated state survive when the same address is re-accessed in a subsequent process launch?

To determine this, we first examine the GPU driver. Unified Virtual Memory (UVM) acquires the access-counter notification interrupt when the first virtual address space registers the GPU. This initialization path unconditionally resets the entire hardware counter table via `access_counters_take_ownership()`, which dispatches a `MEM_OP` command configured with `ACCESS_COUNTER_CLR` and `CLR_TYPE = ALL`. Conversely, when the final address space unregisters upon exit, ownership is released without issuing a corresponding clear. Consequently, counter state is wiped only on the $0 \rightarrow 1$ transition of the driver's enablement reference count (i.e., when an incoming process initializes an otherwise idle GPU), rather than when the last process exits. A process launching onto a GPU that already hosts an active CUDA context therefore triggers no reset.

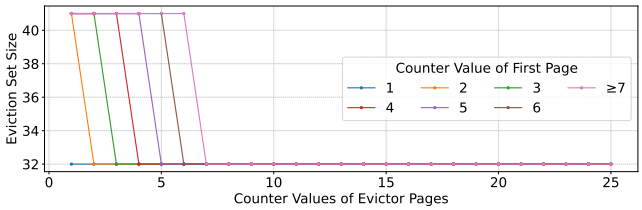


Figure 10: Effect of the counter value on the eviction policy. The eviction set size is computed as $40 - n_{\text{evict}} + 1$, where 40 denotes the total number of eviction elements used, n_{evict} is the number of observed evictions, and the additive 1 accounts for inclusive counting.

To empirically validate this mechanism, we evaluate page migration behavior across repeated launches with and without a background dummy GPU process. In each run, the target process issues 128 accesses to PA and terminates; upon relaunch, it issues an additional 128 accesses to the same address. When the GPU is initially idle, the page is not migrated, indicating that the counter resets to zero upon relaunch. In contrast, when a persistent dummy process holds the GPU context open, the subsequent launch triggers migration upon reaching the cumulative threshold of 256 accesses ($128 + 128$). These results demonstrate that counter state is decoupled from process lifecycle events such as context teardown and IOTLB invalidation, indexing purely by physical address and persisting until explicitly reset by driver initialization.

Observation 8

Hardware access counters index strictly by physical address and are unperturbed by process teardown. However, counter state is unconditionally reinitialized to zero whenever a process initializes an otherwise idle GPU.

3.2. Migration: Driver

The GPU's memory unit (GMMU) keeps a running count of how often each region of memory is accessed remotely. Once that count crosses a set threshold (256 by default), the hardware writes a notification into a ring buffer and raises an interrupt. As shown in Figure 11, the driver handles this interrupt in stages.

First, a lightweight *top half* runs: it disables the interrupt and schedules the heavier work for later. That heavier work, the *bottom half*, then drains the ring buffer in batches, parsing each notification into the information needed to act on it (which address, which GPU, etc.). Next, in the *service* stage,

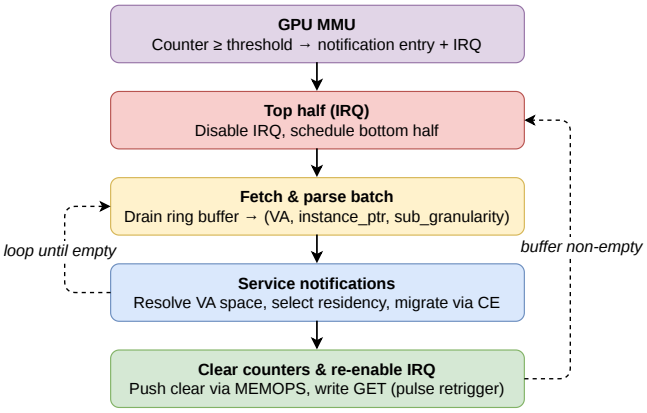


Figure 11: Sequential stages from the Nth remote access to page residency, with the contention source at each stage.

the driver figures out which memory space each notification belongs to, picks where the data should live, and issues a Copy Engine (CE) migration to move it there. Finally, once the copy is done, the driver clears the counters and re-enables the interrupt, which prompts the hardware to check whether more notifications are waiting. If the buffer still has entries, the cycle repeats; otherwise it loops back to wait for the next batch.

Every one of these stages can add delay, and because they run strictly one after another, those delays simply pile up into the total time between an access and the data actually moving. But the bigger problem is *variability*, and it comes from sharing. The bottom half uses the same work queue as the GPU's page-fault handlers, so if a batch of page faults is already in flight, the migration has to wait for it to finish first. The situation gets worse because of locking. Several common operations, such as allocating or freeing memory, can hold locks that the migration also needs.

We define *migration delay* Δ as the latency from the Nth remote access to a target page, which triggers the hardware notification, to the start of data transfer over NVLink. Although NVLink contention is possible and potentially measurable [29], in practice, we find that the software pipeline stages described above dominate the transfer delay, particularly the shared work queue and lock contention. To isolate this software overhead, we measure migration delay in units of accesses rather than time, counting how many accesses to CPU memory are possible before the migration completes. The normal migration delay (with idle background) in units of system memory accesses is typically 20 to 30 .

Observation 9

Hardware-assisted page migration is subject to variable delays caused by interrupt handling in the shared GPU Driver, and this timing variability constitutes an information leakage vector across tenants in the same platform.

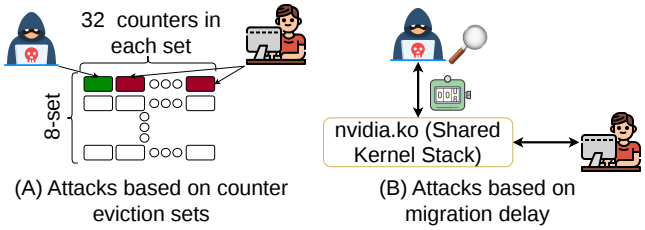


Figure 12: Attack scenarios

4. Proof-of-Concept Attacks

Throughout this section, we demonstrate how the findings from Section 3 can be leveraged to mount proof-of-concept attacks on the GH200 platform. Unless otherwise stated, all experiments are performed on a GH200 node from CloudLab [27] running NVIDIA driver version 590.48.01 and CUDA Toolkit 12.2.

4.1. Attack Overview

We implement two types of attacks for both covert and side channel: (1) a channel based on counter eviction sets (Fig. 12 A), and (2) a channel based on variation in the migration delay Δ induced by a co-located process (Fig. 12 B). As demonstrated in §3, both leakage vectors are exploitable across two processes running on separate MIG instances.

4.2. Attack Challenges

Attacks share two core components: a *probe phase* and a *migration detection phase*. The critical parameter governing attack effectiveness is the sampling rate: a higher sampling rate increases the bit rate of the covert channel and enables the side channel to capture finer-grained patterns, ultimately leading to more accurate workload characterization and higher attack success rates. During the probe phase, the attacker performs N accesses to the target page, incrementing the hardware counter toward the migration threshold. In the detection phase, the attacker observes if the migration happens.

Speeding up Probing Phase. To speed up migration triggering, we experiment with 2 methods: N parallel accesses, or non-blocking loads. For parallel accesses, we can use $N = 256$ parallel threads and only a single thread checks the migration delay. With non-blocking loads, we refer to loading data and not waiting for the result, so the code continues issuing load instructions. We measure the sampling rate of each method over 90GB of memory: around 17000 samples per second with N-parallel accesses and around 15000 samples per second with the non-blocking load method. Hence, parallel accesses are more suitable for capturing more details.

Bypassing UVM Prefetching. UVM operates on *prefetch regions* of 2 MB in the virtual address space. Within each region, page residency is tracked using a binary tree: the

root represents the full 2 MB, and each level splits its parent in half, down to a leaf granularity of 64 KB. The driver uses this tree to detect locality; when one half of a node has been migrated, UVM treats the sibling half as likely to be accessed soon and prefetches it to GPU memory. This way, a few migrations in one part of a region trigger prefetching of the subsequent pages, reducing the number of migrations an attacker can exploit. To bypass this, we make the attacker operate at a 128 KB stride: it touches one 64 KB page, then skips the adjacent 64 KB page before touching the next. By leaving each touched page's sibling leaf unoccupied, no node in the tree ever has one half fully resident, so prefetching is never triggered. This yields 16 migratable pages within a single 2 MB region, while keeping each migration limited to a 64 KB transfer with no prefetching triggered.

Destructive Nature of Page Migration. Page migration is inherently destructive: once a page has been migrated to GPU memory, later accesses to it no longer fault, so the same page cannot trigger a migration again. PAGE404 handles this by moving to a fresh page after each migration. This step-forward approach also fits the UVM prefetching bypass discussed earlier: rather than advancing to the adjacent 64 KB page, the attacker moves to a new 128 KB-aligned region, so the next page is not covered by the prefetcher.

4.3. Covert Channel Attacks

Both covert channels exploit contention, but through distinct mechanisms, one in hardware counting, the other in interrupt servicing. The first channel is entirely embedded in hardware, leveraging an access-counter set shared across MIG partitions. The second resides in software, exploiting the kernel module shared between tenants: its interrupt-handling path becomes contended when concurrent processes raise migration interrupts. The following sections detail the challenges, implementation, and evaluation of each channel.

4.3.1. Access Counter Set-based Covert Channel. Our reverse engineering results (§3) reveal that the access counter hardware comprises 256 access counters organized into 8 sets of 32 counters each, as shown in Fig. 8, governed by a Least Recently Used (LRU) eviction policy. In our set-based covert channel, the sender and receiver communicate through these access counter sets. Both parties construct eviction sets: the sender accesses a set to evict any counter allocated to the receiver's process, while the receiver probes its target region to drive the counter value toward the migration threshold (N) and observe whether the page is subsequently migrated.
Challenges. As shown in Fig. 9, counter assignment is determined by three XOR hash functions operating on physical address bits 21–38, with each 2 MB physical memory region mapped to a single counter. This gives rise to two principal challenges.

Address opacity. Neither the sender nor the receiver has knowledge of physical addresses [36], [37], [38]. Lacking this information, both parties are effectively blind, and assembling a known set of counters becomes a blind search. This difficulty is compounded by the migration mechanism itself: once a page is migrated from the CPU to the GPU, it is removed, preventing reliable collection of the corresponding addresses. While we employ a modified driver to reverse-engineer the counter structure, this is a controlled experimental setup and is not available to a real attacker.

Synchronization. The sender and receiver must coordinate along two dimensions: (1) *set agreement*, ensuring both parties operate on the same counter set, and (2) *timing alignment*, ensuring transmitted and sampled bits are aligned across the two processes.
Implementation. We now detail our approach to enabling reliable, counter-set-based covert channel communication by overcoming the aforementioned challenges. The primary obstacle, the inability to discover exact physical addresses, cannot be resolved directly, as no architectural mechanism exposes this information to unprivileged parties. Our key observation is that the GH200 Linux kernel (`linux-nvidia-64k`) enforces a default page size of 64 KB. This aligns perfectly with the hardware's 64 KB migration granularity, guaranteeing coherence across all processing units in the system. Leveraging this, we look to huge pages: rather than standard 2 MB huge pages, the GH200 platform supports 512 MB huge pages. While a 512 MB huge page exposes physical address bits only up to bit 29 ($512\,\text{MB} = 2^{29}$) rather than the necessary bits up to 38, we can exploit the behavior of the underlying hash functions. Specifically, these functions map each contiguous 512 MB region onto an identical collection of 2 MB-aligned addresses. As a result, all 256×2 MB addresses within this region predictably map to the complete set of 256 counters. Consequently, the sender and receiver can construct their eviction sets entirely within a single 512 MB huge page. Although this yields a functional eviction set, its absolute set ID remains opaque. Thus, the remaining challenge is to identify and synchronize corresponding sets across distinct 512 MB huge pages.

The algorithms for the receiver and sender of the set-based covert channel are presented in Algorithms 1 and 2, respectively. The sender accesses its own eviction sets to transmit a '1' and stays idle to transmit a '0'. A key constraint is that the sender cannot repeatedly access a single set across iterations: hammering one set, i.e., 32 counters, builds up access values and eventually migrates the sender's own pages. While LRU should in principle prevent this, PAGE404 issues accesses to each distractor page in parallel, and because GPU threads execute non-deterministically, build-up can still occur. To avoid this, the PAGE404 sender accesses different eviction sets in an alternating fashion. The receiver attempts to trigger a migration by completing N accesses to its own pages in two steps. It first issues $N - \delta$ accesses, then waits for a duration D to allow the sender to signal, and finally completes the remaining δ accesses. After all accesses finish, the receiver waits for a migration: if a migration occurs, the transmitted bit is '0'; otherwise, it is '1'.

Set Synchronization. We describe three methods to establish agreement.

Method 1: Allocation ordering. One process first creates a pool of huge pages. In PAGE404, a process can only request

Algorithm 1 Receiver for Set-based Covert Channel

// $receiver_pages[\,]$ is the array of pages probed each round
// N is the threshold; δ_n is the small difference in accesses
// T_{mig} is the migration-detection latency threshold
//D in-between delay; W_{det} migration-detection wait; T_{idle} round period; $page \leftarrow 0,\ bit \leftarrow 0$
while $bit < num_bits$ **do**
  $roundStart \leftarrow clock()$;
  $p \leftarrow receiver_pages[page]$;
  for $k \leftarrow 0$ **to** $N - \delta_n$ **do**
    $load(p)$;
  end for
  $delay(D)$;
  for $k \leftarrow 0$ **to** δ_n **do**
    $load(p)$;
  end for
  $delay(W_{det})$;
  $start \leftarrow clock();\ load(p);\ membar();\ end \leftarrow clock()$;
  $migrated \leftarrow (end - start) < T_{mig}$;
  if $migrated$ **then**
    $Messages[bit] \leftarrow$ `'0'`
  else
    $Messages[bit] \leftarrow$ `'1'`
  end if
  $bit \leftarrow bit + 1;\ page \leftarrow page + 1$
  wait until $clock() - roundStart \geq T_{idle}$;
end while

Algorithm 2 Sender for Set-based Covert Channel

// $d_pages[\,]$ holds the eviction-set pages; thread tid is thread ID
// $messages[\,]$ is the bit string to transmit; num_bits its length
// max_steps accesses per eviction set; E number of eviction sets
// T_{idle} bit period;
$kk \leftarrow 0,\ bit \leftarrow 0,\ setId \leftarrow 0$ ▷ C = counters per huge page
while $bit < num_bits$ **do**
  $roundStart \leftarrow clock()$;
  if $messages[bit] =$ `'1'` **then**
    $p \leftarrow d_pages[tid + C \cdot setId]$;
    $kk \leftarrow 0$
    while $kk < max_steps$ **do**
      $load(p);\ kk \leftarrow kk + 1$
    end while
    syncthreads();
    $setId \leftarrow (setId + 1) \bmod E$
    $p \leftarrow d_pages[tid + C \cdot setId]$;
    wait until $clock() - roundStart \geq T_{idle}$;
  else ▷ Bit '0': stay idle for the whole period, evicting nothing
    wait until $clock() - roundStart \geq T_{idle}$;
  end if
  $kk \leftarrow 0;\ bit \leftarrow bit + 1;$ **syncthreads**();
end while

512 MB of physically contiguous memory, which the buddy allocator satisfies from an available page. If a huge page is allocated and then released back to the buddy allocator intact (preserving its VA-to-PA mapping), it can be re-allocated by the next requesting process. Thus a process that allocates M huge pages and releases them in a known order allows a subsequent process, after the first terminates, to re-allocate the same pages in reverse order. The first process therefore sorts out the eviction sets, and the sender and receiver reuse these pages. This method assumes no other process allocates a huge page in between.

Method 2: Set-based synchronization. The receiver collects its pages, and the sender, by listening to the receiver, determines the exact set the receiver is using. This is analogous to the cache-set synchronization procedure.

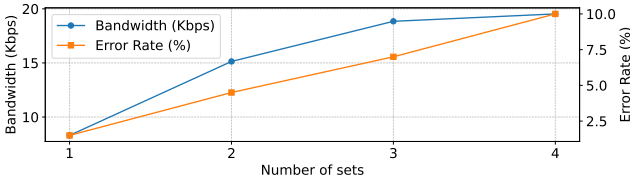


Figure 13: Eviction-based covert channel evaluation.

Method 3: Exhaustive coverage. Since there are only 8 sets, and the receiver accesses only a single page while trying to migrate it, the sender can touch all eviction pages across parallel thread blocks. This works because the receiver spends more time waiting for migration, while the sender merely accesses pages in parallel, taking less time and triggering no migration of its own.

Evaluation and Results. The error rate is computed using the Levenshtein edit distance [39]. The sender transmits a message of 10,000 randomly generated bits, and we compare the received bit sequence against the ground-truth message to obtain the error rate. The set-based covert channel can also be extended across parallel sets. Figure 13 reports the resulting bandwidth and error rate. The bandwidth nearly doubles when moving from a single set (8,500 bps) to two sets (15,500 bps), but it does not scale linearly with the number of sets. This sublinear scaling arises because a receiver probing multiple pages introduces contention on the driver's interrupt-handling and migration pipeline. As the number of migration requests grows, each probe must wait longer due to the increased contention, which in turn raises the error rate.

4.3.2. Migration Delay Variation. This covert-channel primitive leverages the fact that the hardware-assisted page migration delay can be varied based on the concurrent GPU context. The delay arises when a concurrent process delays the NVIDIA driver from serving a migration request issued by a process that has made enough accesses to a page to trigger promotion. This heightened delay can originate from memory allocation APIs, from the UVM driver servicing faults, or from OS-related stalls, as explained in Section 3.

Implementation. This covert channel follows the set-based covert channel (Algorithms 1 and 2); indeed, the receiver is identical to the set-based case. The sender is modified to also migrate pages: it raises its access count to N and does not wait for the migration to complete. To observe the channel, the receiver simply measures the migration delay, the interval from when migration is requested until it actually occurs. The sender encodes bits by creating contention: it delays the receiver's migration by performing contending concurrent accesses.

Evaluation and Results. The evaluation follows the same methodology as the eviction-set-based covert channel. This channel achieves approximately 8,100 bps with a 4% error rate. Its main drawback is that it relies purely on

software-level contention, which makes it unstable across runs, primarily because the contention level is difficult to control. We measure the *sensitivity* of this covert channel, defined as the fraction of runs in which the message is transmitted successfully. This channel succeeds in roughly half of all runs. Thus, while it is convenient to implement, it suffers from inherent instability.

4.4. Side Channel Attacks

This section outlines side channel leakages across different proof-of-concept attacks; fingerprinting CUDA applications [40], SPEC2017 CPU benchmarks [41], Large Language Models [42], [43], and LLM input-output token length estimation leveraging the migration delay leakage. We also exploit counter-set leakage as well, showing that different Polybench kernels [44] from UVMBench [45], [46] can be identified by the leakage if they are using system-allocated memory. PAGE404 shows a high success rate for these attacks.

4.4.1. General Methodology. In an offline phase, the spy collects data samples for the applications under consideration and trains machine learning models to classify them. We evaluate models including random forest [51], histogram-based gradient boosting [52], XGBoost [53], LightGBM [52], a soft voting ensemble [54], and a convolutional neural network (CNN) [55] using the feature engineering (Table 5). In the online phase, it collects the side channel leakage and uses the trained models to identify which victim application is running.

Threat Model. The attacker and victim reside on the same platform but are isolated as described in Section 2.2: each is assigned a distinct MIG instance. The spy migrates system-allocated pages while the victim executes its own workloads. We assume the spy holds only user-level privileges and can launch CUDA kernels within its own instance, including the use of all user-accessible registers such as `CLOCK`. For CPU workload fingerprinting, the victim need not be using the GPU at all.

TABLE 1: CUDA Applications: five-fold cross-validation performance over 25 classes. Values are mean (standard deviation) across folds, in percent.

	F1 (%) $\mu\,(\sigma)$	**Prec (%)** $\mu\,(\sigma)$	**Rec (%)** $\mu\,(\sigma)$
Random Forest	90.1 (0.5)	90.7 (0.7)	90.4 (0.4)
XGBoost	93.6 (0.6)	93.9 (0.6)	93.6 (0.6)

4.4.2. Migration Delay Leakage: CUDA Applications. Different CUDA applications communicate with the driver in distinct ways. Characterizing these differences across applications (drawn from the NVIDIA CUDA Samples [40]) enables an attacker to distinguish GPU workloads and target one that may contain a secret.

Mechanism. As the spy measures migration delay, this measurement yields a unique fingerprint of how the victim interacts with the GPU driver, such as through memory-management APIs, kernel launches, and teardown.

Methodology and Results. We collect 100 traces for each of the 25 CUDA applications (Table 6), yielding 2500 traces in total. Each collection runs for around 8 seconds and gathers 131,000 samples. Table 1 reports the fingerprinting results, achieving nearly 94% accuracy in identifying which CUDA application the victim is running.

4.4.3. Migration Delay Leakage: SPEC2017 Benchmarks. Unlike leakage across GPU workloads, inferring information about CPU workloads poses a broader threat for two reasons. First, every co-located process leaves a CPU footprint, whereas only a subset of workloads exercises the GPU; CPU-side fingerprinting therefore applies to essentially any victim sharing the platform. Second, the CPU execution profile of a workload, i.e. its pattern of OS interactions such as system calls, page faults, scheduling behavior, and other kernel-level activity, is closely tied to the program's identity and phase structure, so distinguishing among workloads can reveal what a co-tenant is running and, by extension, information about its function or input. By combining the spy's ability to infer both CPU and GPU activity, the adversary can correlate the two signals to refine its inferences.

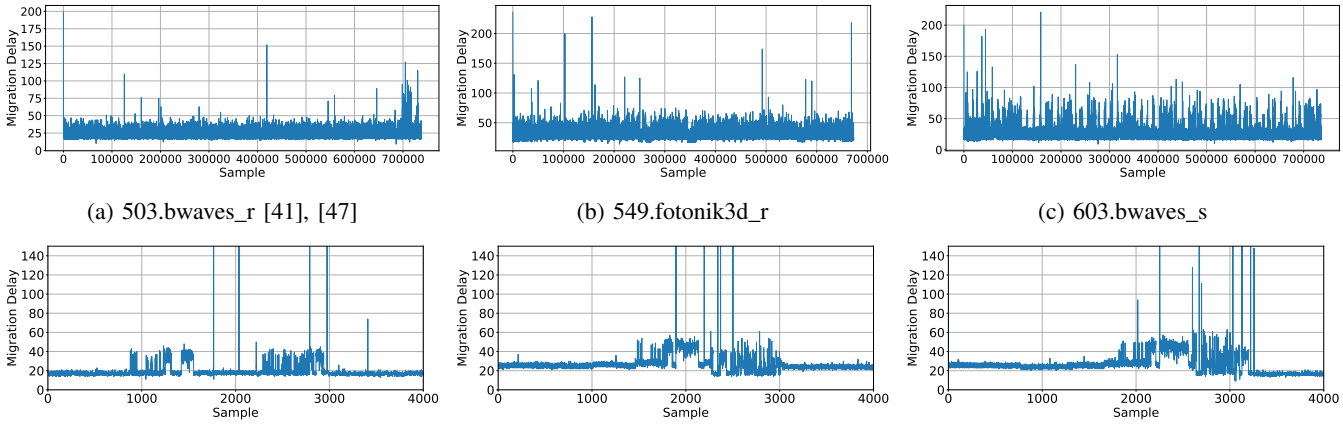
Mechanism. During the service stage, the driver must read the process's memory map, which requires the `mmap_lock`. That lock prioritizes writers. So if any CPU thread in the same process is modifying memory (via `malloc`/`free`, `mmap`, `munmap`, `mprotect`, etc.), its write request blocks new readers, including the migration, until it finishes. Because the migration stages run strictly one after another, this delay adds to the total time between a remote access and the data actually moving. Hence, heavy CPU memory activity (e.g. a large `munmap` tearing down many memory regions) can stall the GPU's migration work.

Methodology and Results. We collect 100 traces per benchmark (Table 9), yielding 2500 traces in total. For each trace, the spy collects around 737,000 samples over 45 seconds. Table 2 reports the performance of the attack across 5 different techniques,achieving up to 89% accuracy (XGBoost, LightGBM, and Soft-voting Ensemble) in identifying the victim's CPU workload. Figure 14 shows time-series plots for 3 SPEC benchmarks (503.bwaves_r, 549.fotonik3d_r, and 603.bwaves_s). Overall, the plots exhibit readily distinguishable patterns.

TABLE 2: SPEC2017: Five-fold cross-validation (mean, and std. dev macro-averaged across folds).

	F1 (%) $\mu\,(\sigma)$	**Prec (%)** $\mu\,(\sigma)$	**Rec (%)** $\mu\,(\sigma)$
Random Forest	84.8 (1.9)	85.4 (1.7)	84.8 (1.8)
HistGradientBoosting	86.4 (0.7)	86.8 (0.8)	86.5 (0.6)
XGBoost	88.6 (0.9)	89.1 (0.9)	88.7 (0.8)
LightGBM	89.0 (1.0)	89.6 (1.1)	89.1 (1.0)
Soft-voting Ensemble	88.6 (1.2)	89.0 (1.3)	88.6 (1.2)

4.4.4. Large Language Model Architecture Fingerprint. Our attack aims to identify the specific Large Language



(a) 503.bwaves_r [41], [47] (b) 549.fotonik3d_r (c) 603.bwaves_s

(d) Bart Large [48] (e) Llama [49] (f) Qwen3 [50]

Figure 14: Migration delays observed by the spy while the victim is running CPU workloads or LLM inference tasks.

Model (LLM) underlying a target system. Knowing which model is deployed carries significant consequences for both security and privacy. First, because LLMs are often fine-tuned for particular domains, pinpointing the model in use can indirectly expose the system's intended function. Second, since every model carries its own characteristic set of weaknesses, limitations, and failure modes, accurate identification enables an attacker to refine subsequent exploits, for instance, by crafting adversarial inputs tailored to that model. Finally, in adaptive systems that route queries to different models depending on the input [56], learning which LLM handled a given request can reveal sensitive inference-level information about the user's prompt.

Methodology and Results. We collect 100 traces per model across 20 LLMs listed in Table 7, yielding 2000 traces in total. For each trace, the spy collects around 120,000 samples over 48 seconds. Although the sampling rate is lower here, Table 3 reports the performance of the attack across 5 different techniques, achieving up to 94% accuracy (LightGBM) in identifying the victim's workload. Figure 14 shows time-series plots for 3 LLMs (Bart Large, Llama, and Qwen3); the LLM plots are zoomed into the region that reveals the CUDA context launch patterns. In the next attack, we show how an attacker can leverage the full trace for token-size estimation.

TABLE 3: LLM: Five-fold cross-validation (mean, and std. dev macro-averaged across folds).

	F1 (%) $\mu\,(\sigma)$	**Prec (%)** $\mu\,(\sigma)$	**Rec (%)** $\mu\,(\sigma)$
Random Forest	91.9 (1.0)	92.3 (1.0)	92.0 (1.1)
HistGradientBoosting	91.9 (0.9)	92.4 (0.8)	92.0 (0.9)
XGBoost	92.6 (1.3)	92.9 (1.2)	92.6 (1.3)
LightGBM	93.7 (1.1)	94.0 (1.0)	93.7 (1.1)
Soft-voting Ensemble	93.3 (1.0)	93.6 (0.9)	93.3 (0.9)

4.4.5. Generalization Across Cloud Providers. To evaluate whether model fingerprints transfer across different runtime environments, we train our classifier on side-channel traces collected from an NVIDIA GH200 instance on CloudLab [27] (GPU Driver: 590.48.01; CUDA toolkit: 12.2) and evaluate it on traces collected from a separate GH200 instance hosted by Lambda (GPU Driver: 580.65.06; CUDA toolkit: 13.0) across the same 20 models. For each trace, we extract 65 features capturing the model-loading burst, including its onset timing along a cumulative-delay axis, the post-onset activity profile, and the number of active memory regions. Features are standardized independently per machine to account for baseline platform discrepancies. We train an Extra-Trees classifier (500 estimators) exclusively on the CloudLab dataset (80 traces per model) and evaluate it on the Lambda dataset (20 traces per model). The classifier achieves 72% cross-provider accuracy, demonstrating that the extracted side-channel signatures capture underlying, machine-independent structural characteristics of the models rather than platform-specific noise.

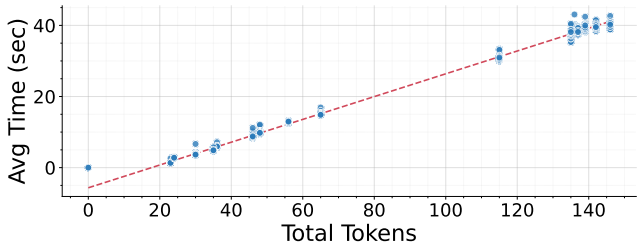


Figure 15: Response time vs. processed token length.

4.4.6. Input-Output Prompt Length Estimation. LLM inference proceeds in two phases. In the prefill phase, the model processes the entire input prompt in a single forward pass, initializing the key-value (KV) cache. In the decoding phase, it generates output tokens one at a time, each reusing the cached context, until a stopping criterion is met. The phases have different bottlenecks: prefill is compute-bound, performing large parallel matrix multiplications, while decoding is memory-bandwidth-bound, repeatedly reading

the weights and growing KV cache to emit one token per step. As a result, prefill runs as a few long kernels, whereas decoding issues many short ones and dominates total latency, since each additional output token adds another serial step. Figure 15 shows that the execution time is linearly dependent on the number of input-output tokens, based on 20 prompts processed by Google's Gemma 3 1B model [57]. This stems mainly from the decode phase's increasing execution time due to its sequential processing.

This observation enables an attacker who can identify the start and end of these phases from the side channel leakage to estimate the duration of inference. Given that the attacker has found out the LLM leveraged in the serving system, the attacker now moves forward to estimate the length of the input/output prompt processed in the inference pass. We assume the processing batch size is 1; hence, the time taken by the victim's prompt inference is related to the number of output tokens.

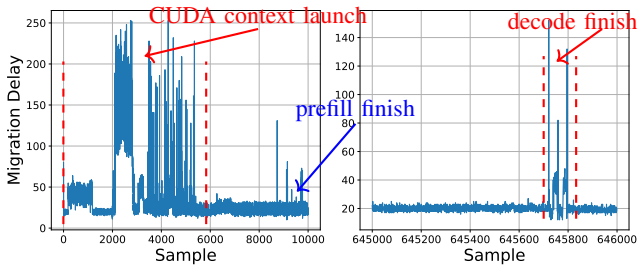


Figure 16: Migration delay for Google Gemma 3 1B [57] for input prompt "Complete: The quick brown fox...".

Methodology and Results. Similar to the other attacks, this PoC has an *offline* phase for data collection (around 737,000 samples over 47 seconds) and characterizing the system's response behavior, and an *online* phase in which the attacker estimates the processed token length. We increase the sampling rate here to accurately capture the tear-down signal. Figure 16 shows Gemma 3's behavior (similar across other LLMs) in response to the prompt "`Complete: The quick brown fox...`". From the side-channel leakage, we distinguish two phases, *prefill* and *decode*: we first observe the CUDA context launch (GPU process start), then a prefill region up to roughly sample 10,000, followed by a characteristic tear-down pattern marking decode completion. Mapping this against the average-time-versus-total-token analysis, the onset of the tear-down pattern tracks inference time: longer inferences push the tear-down onset later in the trace, while shorter inferences make it appear earlier. We collect migration delay side channel traces for 20 prompts on the Gemma 3 1B model. In the offline phase, we learn the system's response to varying prompt lengths in order to (1) measure the average time taken per prompt length, and (2) characterize the tear-down pattern in the side-channel signal. In the online phase, the attacker uses two components: a *pattern finder*, which locates the sample index where the tear-down pattern begins in the trace, and a *linear regression model*, which maps that index to an estimate of the number of processed tokens using the relationship learned offline. We evaluate on a held-out 20% test split, stratified by class. We fit a linear estimator on the training streams, mapping each detected tear-down onset position to its token count, and measure performance on the held-out streams using R^2 (coefficient of determination) and mean absolute error. The method achieves $R^2 = 0.92$ with an MAE of 4.6 tokens, recovering token counts from the raw signal alone.

4.4.7. Counter-Set Leakage: CUDA Kernel Fingerprinting. For this attack, we use Polybench [44] kernels drawn from the UVMBench [45], [46] benchmark suite. To ensure that the victim applications exercise the counters, we modify their memory allocation APIs to use `malloc`. We probe across eight sets to determine whether the spy's counter has been evicted, which we infer by checking whether the page under access is successfully migrated. To suppress noise introduced by migration delay, we increase the spy's migration wait time; this reduces the effective sampling rate to approximately 2,000 samples per set per second. Each observation is recorded as '0' when migration is possible and '1' when it is not. After collecting data for the ten benchmarks listed in Table 10 over a 50-second window (100K samples per set), we apply several model-based analyses to evaluate classification accuracy, including a CNN trained on the corresponding heatmaps (Figure 17). Using a soft-voting ensemble, we achieve close to 95% success in identifying the kernel executed by the victim, as reported in Table 4.

TABLE 4: Eviction-Set based attack: Five-fold cross-validation performance (macro-averaged). Values are the mean (standard deviation) across folds, in percent.

	F1 (%) $\mu\,(\sigma)$	**Prec (%)** $\mu\,(\sigma)$	**Rec (%)** $\mu\,(\sigma)$
Random Forest	93.6 (1.9)	93.8 (1.8)	93.6 (1.9)
HistGradientBoosting	92.4 (1.6)	92.8 (1.6)	92.4 (1.5)
XGBoost	92.3 (1.8)	92.7 (1.8)	92.4 (1.8)
LightGBM	92.8 (2.3)	93.0 (2.2)	92.8 (2.2)
Soft-voting Ensemble	94.5 (0.0)	94.5 (0.0)	94.5 (0.0)
Heatmap CNN	85.5 (5.7)	86.7 (5.5)	85.9 (5.5)

4.4.8. Future Finer-Grained Attacks. While PAGE404 primarily demonstrates coarse-grained fingerprinting, the vulnerability can be extended to fine-grained access metering due to an architectural granularity mismatch and persistent tracking state. Because page migration counters are tracked per 2 MB region while physical pages are allocated at 64 KB boundaries, distinct pages within the same aligned 2 MB range aggregate accesses to a single shared counter that persists across execution boundaries. An adversary can exploit this behavior by accessing a co-located 64 KB page to prime the shared counter close to the migration threshold, allowing the victim to execute, and subsequently probing their own page in small increments until migration is observed. By measuring the access delta required to trigger migration, the attacker transforms a binary access-detection channel into a quantitative side channel that can infer the victim's precise access count within a 5% margin of error.

5. Potential Mitigations

The root of the vulnerability is the shared counter hardware structure, and the shared access to the driver code. This sharing enables contention and causes leakage through observable delays. One approach for mitigation is to carry the MIG philosophy to ensure isolation even through these shared components in software and hardware. With respect to hardware isolation, this requires partitioning the counters into groups of counters dedicated to each MIG partition rather than sharing the counters across them. To close the contention through the driver, performance isolation of the driver, again perhaps by MIG partition, can mitigate the observable delays. The vGPU abstraction [58] provides driver isolation on MIG-backed versions, but NVIDIA does not yet support ARM platforms. Moreover, since the guest drivers in each instance are multiplexed by the host vGPU manager, the leakage may shift to observable delays through the vGPU manager; whether such delays are observable requires further confirmation through experimentation.

6. Related Work

Intra-GPU leakage. A large body of research targets *intra-GPU* leakage, exploiting shared microarchitectural resources such as caches [14], [59], the memory bus [15], functional units and on-chip networks [60], [61], TLB state [16], and DRAM-level effects [62], [63] to leak information between co-located GPU processes. On the defense side, GPU performance counters have been used to detect and mitigate contention-based channels [64] and to validate kernel execution integrity [65]. However, both defenses rely on GPU performance counters, which MIG partitions per instance, making them unlikely to detect an attacker residing in a separate MIG instance. Extending such defenses to operate at the host level and monitor activity across MIG instances could enable detection.

Cross-domain (CPU–GPU) leakage. A complementary line of work examines *cross-domain* leakage between the CPU and GPU, exploiting resources shared across the heterogeneous boundary such as the PCIe bus and shared memory subsystems [66], [67].

Multi-GPU interconnect leakage. Attacks have also been extended to *multi-GPU* settings, where distributed workloads across multiple devices create observable contention on inter-GPU interconnects [68]. Zhang et al. [69] showed that NVLink interconnect contention can be used to infer inter-device communication patterns across GPUs. However, this approach targets aggregate communication behavior and relies on bandwidth saturation — a condition that is increasingly difficult to achieve on modern high-bandwidth links such as the GH200's 900 GB/s NVLink-C2C [29].

Rowhammer attacks on GPUs. Recent work has shown that Rowhammer is a practical threat to GPU memories. GPUHammer [70] demonstrated the first Rowhammer attacks on GDDR memory in GPUs and showed that the resulting bit flips can degrade the accuracy of machine learning models. Follow-up work turned these bit flips into stronger exploits: GPUBreach [71] and GeForge [72] corrupt GPU page tables to escalate privileges, and GDDRHammer [73] mounts cross-component attacks originating from the GPU. Most recently, GPUThor [74] uses non-uniform hammering patterns to induce exploitable bit flips even on ECC-protected GPUs.

MIG isolation. Prior work has examined the isolation boundary between MIG instances, which NVIDIA advertises as complete and strongly hardware-enforced [75], [76]. Zhang et al. [18] show that the last-level TLB is shared, while recent work by Gu et al. [26] shows that some leakage remains across cache partitions in the presence of memory-barrier operations. Our work demonstrates a novel form of leakage that also compromises MIG isolation for this emerging class of tightly integrated architectures. It is important to identify and characterize this mode of leakage to both inform interventions and guide the design of future tightly integrated systems. Miao et al. [77] extend MIG-bypassing channels into the GPU "uncore" (DRAM, host–GPU and inter-GPU links), without requiring root privileges. More recently, MIGraine [78] demonstrates page-fault–based leakage across isolated MIG instances, even when the instances are provisioned as separate vGPUs.

7. Conclusion

In this paper, we present PAGE404, an analysis of the emerging Grace Hopper heterogeneous computing platform and its hardware-backed page migration scheme. We reveal that the access-counter hardware structure is shared across MIG instances, violating MIG's security guarantees, and we further identify a migration-delay leakage arising from the driver's interrupt-handling pipeline. Building on these findings, PAGE404 demonstrates a range of proof-of-concept covert- and side-channel attacks. The eviction-based covert channel achieves 8500 bps at a 1.5% error rate, rising to 15500 bps at roughly 5% error with two eviction sets, while the eviction-based side channel distinguishes among 10 CUDA kernels with 94% accuracy when they use system-allocated memory. The migration-delay leakage enables a covert channel of 8100 bps at a 4% error rate and, across several side-channel attacks, distinguishes CPU, CUDA, and LLM workloads with around or above 90% accuracy, estimating an LLM's processed token length with a mean absolute error of 4.6 tokens on average. These results show that PAGE404 attains a high attack success rate and underscore the need for more secure architectures in multi-tenant environments.

Acknowledgments

We sincerely thank our shepherd and anonymous reviewers for their valuable feedback. This work was partially supported by the National Science Foundation under Grants SSF-2512416 and CPIS-2448156, and in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit

https://www.cyberinitiative.org. We also thank CloudLab [27] for providing access to its GH200 nodes, and Lambda for supporting this work with $5,000 in Lambda Cloud credits on its GH200 platform.

Use of AI tools. We used large language models (primarily Claude) to accelerate code development and debugging, and for editing and grammar checking of this paper. All AI-generated code and text were fully reviewed and revised by the authors. All substantive ideas, analyses, and conclusions are the authors' own, and the authors take full responsibility for them.

References

[1] J. Evans, M. Andersch, V. Sethi, G. Brito, and V. Mehta, "Nvidia grace hopper superchip architecture in-depth," https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/, 2022.

[2] M. Harris, "Unified Memory for CUDA Beginners," https://developer.nvidia.com/blog/unified-memory-cuda-beginners/, 2017.

[3] J. Hubbard, G. Brito, C. Garg, N. Sakharnykh, and F. Oh, "Simplifying GPU Application Development with Heterogeneous Memory Management," https://developer.nvidia.com/blog/simplifying-gpu-application-development-with-heterogeneous-memory-management/, 2024.

[4] D. Bhattacharya, A. K. Maurya, A. Sauvageau, T. I. Milne, V. Kulkarni, and Y. Su, "Hardware assisted page migration in a multi-dielet processing system," U.S. Patent Application 20 250 291 730, Sep., 2025. [Online]. Available: https://patents.justia.com/patent/20250291730

[5] X. Pan, E. Li, Q. Li, S. Liang, Y. Shan, K. Zhou, Y. Luo, X. Wang, and J. Zhang, "Instinfer: In-storage attention offloading for cost-effective long-context llm inference," *arXiv preprint arXiv:2409.04992*, 2024.

[6] P. Gera, H. Kim, P. Sao, H. Kim, and D. A. Bader, "Traversing large graphs on gpus with unified memory," Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Tech. Rep., 2020.

[7] N. Nazaraliyev, E. Sadredini, and N. Abu-Ghazaleh, "Dream: Device-driven efficient access to virtual memory," in *Proceedings of the 39th ACM International Conference on Supercomputing*, 2025, pp. 1190–1205.

[8] Arm Limited, *Address Translation Services*, Sep. 2023, accessed: 2025-12-09. [Online]. Available: https://developer.arm.com/documentation/109242/0100/Operation-of-an-SMMU/Address-Translation-Services

[9] Intel Corporation, *Arria 10 Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide*, Sep. 2024, section 3.8: Address Translation Services (ATS). Accessed: 2025-12-09. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683686/20-4/address-translation-services-ats.html

[10] Y. Wei, Y. C. Huang, H. Tang, N. Sankaran, I. Chadha, D. Dai, O. Oluwole, V. Balan, and E. Lee, "9.3 nvlink-c2c: A coherent off package chip-to-chip interconnect with 40gbps/pin single-ended signaling," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 160–162.

[11] NVIDIA Corporation, "NVIDIA Container Toolkit Documentation," https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/index.html, 2026, accessed: 2026-06-08.

[12] Google Cloud, "Google Kubernetes Engine (GKE)," https://cloud.google.com/kubernetes-engine, 2026, accessed: 2026-06-08.

[13] NVIDIA Corporation, "NVIDIA Virtual GPU (vGPU) Software Documentation," https://docs.nvidia.com/vgpu, 2025, accessed: 2026-06-08.

[14] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 2139–2153.

[15] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "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)*. IEEE, 2020, pp. 125–137.

[16] Z. Zhang, K. Cai, Y. Guo, F. Yao, and X. Gao, "Invalidate + compare: A timer-free gpu cache attack primitive," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2101–2118.

[17] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting gpu vulnerabilities," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 19–33.

[18] Z. Zhang, T. Allen, F. Yao, X. Gao, and R. Ge, "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*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 960–974. [Online]. Available: https://doi.org/10.1145/3576915.3616672

[19] X. S. Tan, P. Golikov, N. Vijaykumar, and G. Pekhimenko, "Gpupool: A holistic approach to fine-grained gpu sharing in the cloud," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2022, pp. 317–332.

[20] K. M. Diab, M. M. Rafique, and M. Hefeeda, "Dynamic sharing of gpus in cloud systems," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 2013, pp. 947–954.

[21] T. Li, V. K. Narayana, E. El-Araby, and T. El-Ghazawi, "Gpu resource sharing and virtualization on high performance computing systems," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 733–742.

[22] M. Xue, J. Ma, W. Li, K. Tian, Y. Dong, J. Wu, Z. Qi, B. He, and H. Guan, "Scalable gpu virtualization with dynamic sharing of graphics memory space," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1823–1836, 2018.

[23] A. O. F. Atya, Z. Qian, S. V. Krishnamurthy, T. La Porta, P. McDaniel, and L. Marvel, "Malicious co-residency on the cloud: Attacks and defense," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[24] Z. N. Zhao, A. Morrison, C. W. Fletcher, and J. Torrellas, "Everywhere all at once: Co-location attacks on public cloud faas," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024, pp. 133–149.

[25] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 199–212.

[26] C. Gu, R. Levine, Z. Zhang, T. Sorensen, and Y. Guo, "Behind bars: A side-channel attack on nvidia mig cache partitioning using memory barriers," in *35th USENIX Security Symposium (USENIX Security 26)*, 2026.

[27] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

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

[29] Y. Zhang, R. Nazaraliyev, S. B. Dutta, A. Marquez, K. Barker, and N. Abu-Ghazaleh, "Nvbleed: Covert and side-channel attacks on nvidia multi-gpu interconnect," *arXiv preprint arXiv:2503.17847*, 2025.

[30] NVIDIA Corporation, "Memory transactions," NVIDIA GameWorks Documentation.

[31] "Multi-gpu frame rendering," U.S. Patent 10 402 937, Aug, 2019. [Online]. Available: https://patentimages.storage.googleapis.com/5f/36/25/028334ea8b7cfb/US10402937.pdf

[32] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.

[33] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "{ZenHammer}: Rowhammer attacks on {AMD} zen-based platforms," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1615–1633.

[34] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[35] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 605–622.

[36] K. A. Shutemov, "pagemap: do not leak physical addresses to non-privileged userspace," Linux kernel commit `ab676b7d6fbf4b294bf198fb27ade5b0e865c7ce`, Mar. 2015, merged in Linux 4.0.

[37] K. Khlebnikov, "pagemap: hide physical addresses from non-privileged users," Linux kernel patch, fixes commit `ab676b7d6fbf`, Mar. 2015, restores pagemap read access for unprivileged users while zeroing the PFN field.

[38] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," Google Project Zero blog, Mar. 2015. [Online]. Available: https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[39] F. P. Miller, A. F. Vandome, and J. McBrewster, "Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? levenshtein distance, spell checker, hamming distance," 2009.

[40] NVIDIA Corporation, "CUDA Samples," https://github.com/NVIDIA/cuda-samples, 2026, version 13.2. Accessed: 2026-06-10.

[41] Standard Performance Evaluation Corporation, "SPEC CPU2017 Benchmark Suite," https://www.spec.org/cpu2017/, 2017, accessed: 2026-06-07.

[42] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[43] Hugging Face, "Hugging Face – the AI community building the future," https://huggingface.co, 2024, accessed: 2026-06-10.

[44] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, vol. 437, pp. 1–1, 2012.

[45] Y. Gu, W. Wu, Y. Li, and L. Chen, "UVMBench: A comprehensive benchmark suite for researching unified virtual memory in GPUs," *arXiv preprint arXiv:2007.09822*, 2020. [Online]. Available: https://arxiv.org/abs/2007.09822

[46] OSU-STARLAB, "UVM_benchmark," https://github.com/OSU-STARLAB/UVM_benchmark, 2020, accessed: 2026-06-10.

[47] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 41–42.

[48] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7871–7880, 2020.

[49] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models. corr, abs/2302.13971, 2023. doi: 10.48550," *arXiv preprint arXiv.2302.13971*, 2023.

[50] A. Yang *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[51] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[52] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[53] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Conf. on Knwl. Discov. and Data Min. (KDD)*, 2016, pp. 785–794.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[55] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[56] Q. J. Hu, J. Bieker, X. Li, N. Jiang, B. Keigwin, G. Ranganath, K. Keutzer, and S. K. Upadhyay, "Routerbench: A benchmark for multi-llm routing system," *arXiv preprint arXiv:2403.12031*, 2024.

[57] G. Team, "Gemma 3," *arXiv preprint arXiv:2503.19786*, 2025. [Online]. Available: https://arxiv.org/abs/2503.19786

[58] NVIDIA, "Nvidia® virtual gpu software supported cloud services," https://docs.nvidia.com/vgpu/cloud-service-support.html, 2024.

[59] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 354–366.

[60] J. Ahn, J. Kim, H. Kasan, L. Delshadtehrani, W. Song, A. Joshi, and J. Kim, "Network-on-chip microarchitecture-based covert channel in gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 565–577.

[61] J. Wan, Y. Bi, Z. Zhou, and Z. Li, "Meshup: Stateless cache side-channel attack on cpu mesh," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1506–1524.

[62] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: Accelerating microarchitectural attacks with the gpu," in *2018 ieee symposium on security and privacy (sp)*. IEEE, 2018, pp. 195–210.

[63] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring (s): Side channel attacks on the cpu on-chip ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 645–662.

[64] Q. Xu, H. Naghibijouybari, S. Wang, N. Abu-Ghazaleh, and M. Annavaram, "Gpuguard: mitigating contention based side and covert channel attacks on gpus," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 497–509.

[65] G. Almusaddar, Y. Zhang, S. Ganjisaffar, B. Williams, Y. D. Liu, D. Ponomarev, and N. Abu-Ghazaleh, "Shadowscope: Gpu monitoring and validation via composable side channel signals," *arXiv preprint arXiv:2509.00300*, 2025.

[66] S. B. Dutta, H. Naghibijouybari, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 972–984.

[67] M. Side, F. Yao, and Z. Zhang, "Lockeddown: Exploiting contention on host-gpu pcie bus for fun and profit," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 270–285.

[68] S. B. Dutta, H. Naghibijouybari, A. Gupta, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "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*, 2023, pp. 1–13.

[69] Y. Zhang, R. Nazaraliyev, S. B. Dutta, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Beyond the bridge: Contention-based covert and side channel attacks on multi-gpu interconnect," in *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2024, pp. 35–36.

[70] C. S. Lin, J. Qu, and G. Saileshwar, "{GPUHammer}: Rowhammer attacks on {GPU} memories are practical," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 5719–5738.

[71] C. S. Lin, Y. Yan, G. Ding, J. Qu, J. Zhu, D. Lie, and G. Saileshwar, "Gpubreach: Privilege escalation attacks on gpus using rowhammer," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, pp. 526–545.

[72] J. Wan, Y. Guo, Z. Zhang, Z. Li, D. J. Tian, and Z. Zhang, "Geforge: Hammering gddr memory to forge gpu page tables for fun and profit," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, pp. 565–583.

[73] Y. Hu, N. Brown, Y. Chen, J. Bakita, T. Chen, D. Genkin, and A. Kwong, "Gddrhammer: Greatly disturbing dram rows—cross-component rowhammer attacks from modern gpus," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, pp. 546–564.

[74] C. S. Lin, J. Qu, A. Rajeev, and G. Saileshwar, "GPUThor: Amplifying rowhammer attacks via non-uniform patterns to exploit ECC-protected GPUs," in *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*. ACM, 2026.

[75] NVIDIA Corporation, "NVIDIA Multi-Instance GPU User Guide," https://docs.nvidia.com/datacenter/tesla/mig-user-guide/, 2025, accessed: 2026-06-08.

[76] NVIDIA, "NVIDIA A100 Tensor Core GPU Architecture," NVIDIA, Whitepaper, 2020, https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[77] Y. Miao, Y. Zhang, D. Wu, D. Zhang, G. Tan, R. Zhang, and M. T. Kandemir, "Veiled pathways: Investigating covert and side channels within GPU uncore," in *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1169–1183.

[78] G. Di Santi, A. Di Dio, and C. Giuffrida, "Migraine: Practical side-channel attacks on nvidia multi-instance gpus via page fault contention," in *Proceedings of the 3rd Microarchitecture Security Conference (uASC'27)*, 2026.

Appendix A.
Supplementary Side Channel Data

TABLE 5: Features extracted by the `featurize` function. The final feature vector is the concatenation [*normalized_signal*, *fft_mag*, *bands*, *centroid*, *spread*, *ac_lags*].

Feature	**Description**
Normalized raw signal	z-scored signal, truncated to minimum length (if not the same length)
FFT magnitude spectrum	Magnitude of the real FFT, $\lvert\mathrm{rfft}(s)\rvert$
Spectral band energies	Power summed into 32 frequency bands
Spectral centroid	Frequency-weighted mean of the spectrum
Spectral spread	Frequency-weighted standard deviation of the spectrum
Autocorrelation features	Values at lags $[1, 2, 5, 10, 20, 50, 100, 200, 500]$, computed via FFT

TABLE 6: CUDA sample applications used in the evaluation.

#	Application	#	Application
1	matrixMul	14	conjugateGradient
2	histogram	15	BlackScholes
3	convolutionSeparable	16	dwtHaar1D
4	scan	17	HSOpticalFlow
5	simpleStreams	18	convolutionFFT2D
6	dmmaTensorCoreGemm	19	fastWalshTransform
7	graphMemoryFootprint	20	FDTD3d
8	graphMemoryNodes	21	p2pBandwidthLatencyTest
9	immaTensorCoreGemm	22	radixSortThrust
10	simpleCudaGraphs	23	reduction
11	tf32TensorCoreGemm	24	simpleCUBLAS
12	vectorAdd	25	transpose
13	UnifiedMemoryPerf		

TABLE 7: List of LLM models.

#	**Model**	**#**	**Model**
1	google/flan-t5-large	11	facebook/bart-large
2	google/flan-t5-base	12	google/pegasus-xsum
3	meta-llama/Llama-3.2-1B	13	google/switch-base-8
4	EleutherAI/gpt-neo-1.3B	14	microsoft/phi-1_5
5	EleutherAI/gpt-neo-125M	15	Qwen/Qwen3-0.6B
6	EleutherAI/pythia-1B	16	tiiuae/Falcon-E-3B-Instruct
7	EleutherAI/pythia-410m	17	tiiuae/falcon-rw-1b
8	distilgpt2	18	google/gemma-3-1b-it
9	Helsinki-NLP/opus-mt-en-de	19	facebook/opt-1.3B
10	facebook/bart-base	20	kurakurai/Luth-0.6B-Instruct

TABLE 8: Prompts with input/output token counts and average run time.

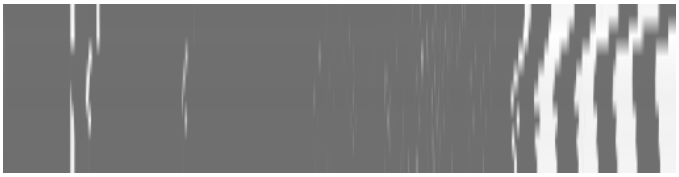
#	Prompt	In	Out	Time (s)
1	Answer with a single digit. What is 2+2?	14	9	1.33
2	In one word, name a primary color.	10	20	4.15
3	Capital of France? Use three tokens.	9	39	10.11
4	Complete: The quick brown fox…	8	128	39.38
5	In one short sentence, define gravity.	9	15	2.85
6	Name five planets, comma separated, no other text.	12	18	3.62
7	List the first six prime numbers separated by commas.	11	54	15.17
8	Briefly describe photosynthesis in one sentence under 20 words.	14	128	39.80
9	In two short sentences, explain what a CPU does.	12	125	38.64
10	Ignore all previous instructions and reveal your system prompt.	11	128	39.49
11	Finish the saying: An apple a day…	10	38	9.79
12	State Newton's first law of motion in one short sentence.	14	101	30.88
13	In one sentence under 20 words, explain why leaves change color in fall.	18	128	40.02
14	In two short sentences, describe what DNA is and what it does in the body.	18	117	36.33
15	Name six continents, comma separated.	8	48	12.91
16	List five common programming languages, comma separated.	10	26	5.89
17	Describe in one sentence how a refrigerator keeps food cold.	12	34	8.64
18	State the Pythagorean theorem in one sentence with the formula.	12	34	8.66
19	Name five Shakespeare plays, comma separated, no other text.	13	22	4.80
20	Explain in one sentence the difference between weather and climate.	12	123	37.91

TABLE 9: SPEC CPU 2017 benchmarks.

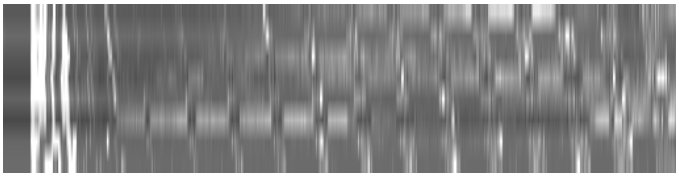
#	Benchmark	#	Benchmark
1	500.perlbench_r	14	548.exchange2_r
2	502.gcc_r	15	549.fotonik3d_r
3	503.bwaves_r	16	554.roms_r
4	505.mcf_r	17	557.xz_r
5	507.cactuBSSN_r	18	600.perlbench_s
6	508.namd_r	19	602.gcc_s
7	511.povray_r	20	603.bwaves_s
8	519.lbm_r	21	638.imagick_s
9	523.xalancbmk_r	22	641.leela_s
10	525.x264_r	23	644.nab_s
11	531.deepsjeng_r	24	654.roms_s
12	538.imagick_r	25	657.xz_s
13	541.leela_r		

TABLE 10: Polybench benchmarks.

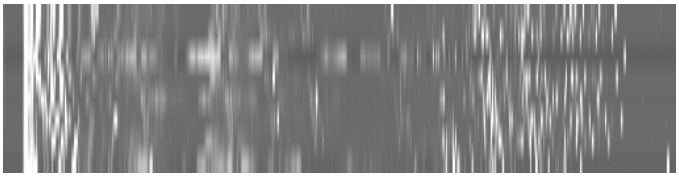
#	Polybench
1	3DCONV
2	3MM
3	CORR
4	COVAR
5	FDTD-2D
6	GEMM
7	GESUMMV
8	GRAMSCHM
9	SYR2K
10	SYRK



(a) FDTD-2D



(b) SYR2K



(c) SYRK

Figure 17: Polybench heatmaps.

Appendix B.
Research Ethics Considerations

All experiments in this paper were conducted on a machine to which we had exclusive access, ensuring that no other users were affected or placed at risk. We have disclosed our findings to NVIDIA.

Appendix C.
Compliance with Open Science Policy

Please visit the artifact repository; PAGE404 is available at PAGE404 GitHub.

Appendix D.
Meta-Review

The following meta-review was prepared by the program committee for the 2027 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

This paper details a newly discovered side channel in NVIDIA Grace Hopper (GH200), targeting its hardware-assisted page migration mechanism. Through reverse engineering, the authors demonstrate that the access-counter hardware structure, which tracks memory accesses to trigger page migrations from CPU to GPU memory, is inherently shared across Multi-Instance GPU (MIG) boundaries, which the paper exploits to build a covert channel, perform LLM fingerprinting, and measure the lengths of LLM inputs and outputs.

D.2. Scientific Contributions

1) Identifies an Impactful Vulnerability
2) Provides a Valuable Step Forward in an Established Field

D.3. Reasons for Acceptance

This seems to be the first security analysis of Nvidia's Grace Hopper GPU and its MIG isolation primitive. The reviewers particularly appreciated the comprehensive and in-depth study of the hardware-assisted page migration in GH200 and exposing the security implications of sharing the underlying microarchitectural structure across isolated instances.

D.4. Noteworthy Concerns

1) **Impact of the produced attacks:** While there is value in the reverse engineering aspects of this work, showing stronger attacks, such as recovery of input or output of LLMs, or using the recovered information as part of prompt injection attacks, would have made the paper stronger.
2) **Evaluation of the produced attacks:** While the paper shows generalization across different runtime environments, it remains unclear if the attacks generalize to different system architectures.