RekSai: Covert Channel Attacks via GPU Last-Level Caches

Lingfeng Pei
lpei@u.nus.edu
National University of Singapore
Singapore, Singapore

Changxi Liu
changxiliu@hkust-gz.edu.cn
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China

Xiaoyu Cheng
xiaoyu_cheng@seu.edu.cn
Southeast University
Nanjing, China

Trevor E. Carlson
tcarlson@nus.edu.sg
National University of Singapore
Singapore, Singapore

Abstract

Graphics Processing Units (GPUs) are central to modern machine learning and high-performance computing workloads, and multi-tenant use in cloud and shared systems has made GPU security a growing concern. In this work, we uncover a previously unknown leakage of control state associated with the GPU L2 last-level cache (LLC), where the effective L2 fetch-granularity state can remain visible across processes sharing the same physical GPU. This behavior shows that existing software- and hardware-level mechanisms, including Multi-Instance GPU (MIG), do not fully isolate this microarchitectural state.

Building on this observation, we introduce RekSai, a covert-channel methodology that leverages the GPU LLC to enable two colluding unprivileged processes to communicate across intended software isolation boundaries. We evaluate RekSai on NVIDIA A100 (Ampere) and H100 (Hopper) GPUs and demonstrate reliable communication across process boundaries on A100 and across MIG instances on both platforms in our evaluated settings. To our knowledge, this is the first covert channel based on leakage through L2 fetch-granularity control state. Our results reveal a gap between software-visible isolation and hardware-visible microarchitectural state in multi-tenant GPUs.

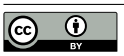
CCS Concepts

• **Security and privacy → Hardware attacks and countermeasures**; *Hardware reverse engineering*.

Keywords

GPU security, covert channels, cache timing attacks, last-level cache, microarchitectural leakage, Multi-Instance GPU (MIG)

ACM Reference Format:
Lingfeng Pei, Changxi Liu, Xiaoyu Cheng, and Trevor E. Carlson. 2026. RekSai: Covert Channel Attacks via GPU Last-Level Caches. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26), November 08–12, 2026, San Jose, CA, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3831252.3833943



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICCAD '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3833943

1 Introduction

Graphics Processing Units (GPUs) have emerged as the primary hardware accelerators for large-scale AI training and other high-performance workloads. Modern datacenters and supercomputers widely deploy GPUs to accelerate a broad range of applications. In particular, NVIDIA GPUs based on the A100 (Ampere) and H100 (Hopper) architectures have seen widespread adoption. Major cloud providers, including AWS, Azure, Google Cloud, and Alibaba Cloud, as well as supercomputers like Eagle [21] and Leonardo [25], rely on NVIDIA GPUs to support advanced scientific and artificial intelligence (AI) research.

To support safe multi-tenant sharing of hardware, modern GPUs provide both software- and hardware-level isolation mechanisms to separate concurrent users and processes. In NVIDIA's software stack, CUDA contexts provide per-process execution environments, while mechanisms such as Multi-Instance GPU (MIG) introduce hardware partitioning across tenants.

CUDA contexts isolate per-process execution and memory spaces, preventing direct access to another process's GPU memory [20]. However, CUDA-context isolation alone does not fully partition all microarchitectural resources, and co-running processes may still share components such as caches, SMs, and on-chip interconnects. MIG further partitions these resources across GPU instances, providing stronger isolation than software-only sharing. Together, these mechanisms are intended to isolate execution and memory resources across tenants and to avoid explicit communication paths across such boundaries.

However, we show that this isolation does not fully extend to the hardware-visible state controlling L2 fetch granularity. Through reverse engineering of the A100 and H100 GPU microarchitecture, including analysis of L2/DRAM access paths and runtime granularity-setting mechanisms, we uncover an effective L2 fetch-granularity control state whose architectural effects are not always confined to the software or hardware partition intended to isolate tenants. In particular, we find that this state can remain visible across distinct execution domains, allowing one user-level process to influence the L2 behavior observed by another. Moreover, this leakage is not limited to cross-process settings: in our evaluated systems, it can also persist across different MIG instances on both the A100 and H100. These results indicate that CUDA-context isolation and MIG-based partitioning do not fully isolate this hardware-visible control state, because its effects remain visible across these boundaries.

In this paper, we present RekSai[1][2], a covert-channel methodology that exploits this control-state leakage on modern NVIDIA GPUs, evaluated on both A100 and H100, to enable communication across intended isolation boundaries. Specifically, we identify a setting exposed through a CUDA runtime/device API that controls L2 cache fetch granularity. By manipulating this setting from an unprivileged process, a sender can change how memory accesses populate L2 cache lines, while a receiver on the same GPU hardware can recover the induced changes through cache timing measurements. Unlike prior GPU covert channels based on contention in resources such as NVLink [5, 29], the on-chip network [1], or the last-level TLB [30], RekSai exploits a deterministic architectural effect that directly changes L2 sector fetch behavior. As a result, the channel is straightforward to realize, operates entirely from unprivileged user space, and remains effective across process boundaries and, in our evaluated settings, across MIG instances as well.

In summary, this paper makes the following contributions:

- We uncover previously undocumented behavior in how Ampere- and Hopper-generation GPUs manage L2 fetch granularity.
- We find that the effective hardware state controlling L2 fetch granularity is not fully isolated by the intended execution boundary. In particular, the effects of this state can remain visible across processes on A100 and across different MIG instances on both A100 and H100, allowing one unprivileged execution domain to influence the L2 cache behavior observed by another.
- We build RekSai, a fully unprivileged covert channel on modern NVIDIA GPUs that is, to our knowledge, the first to exploit L2 fetch-granularity leakage and achieves deterministic, high-quality communication across process boundaries on A100 and, in our evaluated settings, across MIG isolation boundaries on both A100 and H100, revealing that existing process- and partition-level isolation mechanisms do not fully isolate this microarchitectural device state.

2 Background and Related Works

We introduce the GPU architecture and related GPU security research in this section.

GPU Architecture. Modern GPUs feature hierarchical structures in both hardware and software. Typically, a GPU consists of multiple Streaming Multiprocessors (SMs), where each thread block is scheduled on one SM. All warps within a thread block share the same shared memory and L1 cache. Each SM contains a set of GPU cores, and all threads within a warp execute in lockstep, sharing the same program counter (PC). In NVIDIA Ampere GPUs, the L1 cache, texture cache, and shared memory share a unified on-chip data cache, with the shared-memory carveout configurable at runtime.

The L2 cache is shared by all SMs and serves as the last-level cache (LLC). Modern GPUs can have a large L2 cache. For example, the NVIDIA H100 contains a 50 MiB L2 cache. The LLC interfaces with off-chip global memory. Data-center GPUs such as the A100 and H100 use high-bandwidth memory (HBM), whereas many consumer GPUs use graphics double data rate (GDDR) memory. The L2 cache adopts a sectored cache design [22], where each cache line contains multiple sectors (four on Hopper GPUs). All sectors share a common tag, and their addresses are contiguous. To enhance performance across diverse workloads, NVIDIA provides APIs that allow dynamic configuration of cache access granularity, enabling the GPU to fetch small data blocks (32B) for irregular applications to save bandwidth, or large data blocks (128B) for regular applications to maximize throughput.

Unlike conventional I/O or networking devices, GPUs maintain an independent memory hierarchy that operates outside the control of the operating system. Additionally, GPUs isolate the execution of processes through CUDA contexts. Each process running on the GPU has its own CUDA context, which should remain isolated from others during runtime. When multiple processes share a single GPU, the GPU typically executes kernels from one process before scheduling tasks from another. However, to improve utilization, the GPU driver may interleave kernels from different processes when one process has a long idle interval between kernel launches. Consequently, kernels from different processes can execute interleaved on the same GPU.

To improve hardware utilization, NVIDIA introduced the Multi-Instance GPU (MIG) with the NVIDIA A100 (Ampere architecture), and it is supported in subsequent architectures, such as the Hopper architecture. MIG is a hardware-level partitioning technology and allows one physical GPU to be divided into multiple independent *GPU instances*. On A100, a device can be partitioned into up to seven instances, each with dedicated compute and memory resources.

Using A100 as an example, the GPU is organized into seven *GPU Processing Clusters* (GPCs) and partitioned L2-cache and memory resources. Each GPU instance is built from one or more GPU slices, where each slice contains one GPC and one L2 slice group. Each L2 slice group contains 10 L2 cache slices, and each L2 cache slice is associated with one memory controller. This design gives each instance an isolated compute and memory domain, so applications in one instance cannot access the hardware assigned to another instance [18].

MIG is designed to provide hardware-enforced isolation for multiple tenants. Compared with software-only sharing mechanisms, MIG provides stronger isolation. In particular, requests to off-chip DRAM from each instance are restricted to its assigned L2 slices, memory controllers, and DRAM address range, while compute resources are statically partitioned across instances. However, this isolation is not absolute at every microarchitectural layer. We observe that some global GPU components can still induce residual cross-instance interactions. These remaining interactions motivate this work to build cross-instance covert channels.

Cache Timing Attacks. Cache-timing attacks exploit measurable timing differences that arise from microarchitectural cache states. Typically, an attack follows three phases: (1) the attacker prepares a known state for one or more cache lines, (2) the victim performs secret-dependent memory accesses that change those cache states, and (3) the attacker probes the same lines and decodes the secret from the observed timing differences. Any change to cache state, such as occupancy (cache line present or absent) [2, 3, 6, 13, 27, 28], coherence (dirty/shared state) [4, 8, 24], or replacement metadata [26], can be used to encode information. The most common instantiation is occupancy-based: attackers either evict

[1] RekSai is publicly available at https://github.com/lingfengpei/RekSai_Artifact.git

[2] **Responsible Disclosure.** We disclosed RekSai to the NVIDIA Security Team and received acknowledgment under ticket #6188463.

a line using an instruction like the CPU instruction `clflush` (e.g., Flush+Reload [28]) or priming the cache set occupied by the victim (e.g., Prime+Probe [14]); if the victim subsequently loads the evicted line, the attacker observes a faster access, thus yielding a secret-dependent timing difference.

GPU Security. With the increasing importance of GPUs in modern computer systems, GPU security has recently received increasing attention from the research community. *Spy in the GPU-box* [5] builds an attack model based on the observation that memory locations are cached only on the GPU where the memory is allocated, particularly in the Pascal and Volta architectures. *Invalidate+Compare* [31] proposes a cache-based side-channel attack that exploits the properties of the GPU cache hierarchy. Zhixian et al. [12] construct an attack model leveraging the characteristics of the GPU network-on-chip (NoC). *TunneLs for Bootlegging* [30] constructs a cross-MIG covert channel using last-level TLB contention on A30 and A100 GPUs and reports higher raw throughput than RekSai. *Behind Bars* [7] introduces Membar+Load, a cross-MIG covert- and side-channel primitive in which memory barriers from one MIG instance propagate across L2 cache partitions and affect load timing in another instance. In addition, several microarchitectural characterization and reverse-engineering studies [9–11, 15, 23] provide valuable insights into internal GPU designs, which can further aid in building more attack models. However, most of these studies are still based on older architectures, such as Volta, Turing, or even Kepler, which lag behind modern GPU architectures by nearly a decade.

3 GPU Reverse Engineering

In this section, we describe the sector-timing methodology used to reverse engineer L2 fetch granularity and its cross-context control behavior.

Measuring L2 Cache Latency. We adapt the methodology of Mei et al. [16] to measure L2 fetch granularity at runtime. Since both Ampere and Hopper use a NUMA (Non-Uniform Memory Access) organization [18, 19], memory access latency depends on NUMA node locality. To obtain stable measurements, we fix the streaming multiprocessor (SM) ID and launch kernels on the same SM, isolating L2 cache behavior from NUMA variation.

Our measurements reveal two DRAM-latency clusters on both architectures, consistent with NUMA affinity: on A100, local and remote DRAM accesses take about 380 and 550 cycles, respectively, while on H100 they take about 550 and 700 cycles. In contrast, L2 cache latency remains below 300 cycles on both architectures, providing a clear threshold for distinguishing L2 hits from DRAM accesses.

Measuring L2 Fetch Granularity. To infer L2 fetch granularity, we first access a target address and then probe offsets at +32 and +64 bytes. Because the hardware exposes only three settings, 32, 64, and 128 bytes, these two probes are sufficient. If the +32 access incurs DRAM latency, only the first 32-byte sector was fetched. If +32 hits in the L2 but +64 misses, the fetch granularity is 64 bytes. If both +32 and +64 hit in the L2, the fetch granularity is 128 bytes.

L2 Granularity Control Mechanism. To understand how GPUs control this granularity, we interleave kernel execution between a receiver and a sender. The sender invokes `cudaDevice`

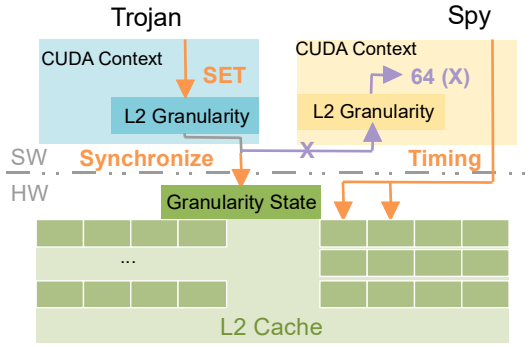


Figure 1: Attack overview. The Trojan and Spy execute on separate GPU instances, whose cache slices are isolated by MIG partitioning. The Trojan process changes its L2 fetch-granularity setting, changing how the GPU's L2 cache fetches cache lines. Although the Spy runs in a separate CUDA context and assumes the default 64-byte granularity, the effective state reflects the setting selected by the Trojan. By timing accesses to selected L2 sectors, the Spy observes Trojan-induced hit/miss patterns and decodes the transmitted data bit.

`SetLimit` with both supported and unsupported values, followed by `cudaDeviceSynchronize`; both calls return `cudaSuccess` in all tested settings. The sender-side `cudaDeviceGetLimit` reports the most recently attempted value, including unsupported values. In the receiver's independent CUDA context, however, `cudaDeviceGetLimit` remains at the default value of 64 bytes throughout these tests. The receiver's L2 sector timings instead reveal the sender-controlled effective behavior: they follow the 32-, 64-, and 128-byte patterns for supported values and the 64-byte pattern for the unsupported values we tested.

$$\text{CUDA Context} \xrightarrow{\text{Set}} \text{Effective Control State} \xrightarrow{\text{Control}} L2 \quad (1)$$

These results strongly suggest a two-level control mechanism. Each process maintains its own L2 fetch-granularity setting in its CUDA context, which is what `cudaDeviceGetLimit` returns. The software-visible value therefore does not necessarily describe the effective granularity observed through timing. However, we discover that the effective L2 fetch behavior observed across MIG instances reflects the control state selected by the most recent call to `cudaDeviceSetLimit`, regardless of which instance issued the call. Because the effect of this state can remain visible across execution boundaries, one process may observe L2 behavior induced by another, forming the basis of our covert channel.

Process Switching. For processes within the same GPU instance, Ampere and Hopper behave differently. In our Ampere environment, the effective L2 fetch-granularity state can remain visible after a process terminates, and a later process can observe behavior induced by the earlier setting. On Hopper, in contrast, we find that the observed behavior follows the CUDA-context setting at each process switch, preserving a consistent per-process setting.

Table 1: Observed platform behavior of the effective L2 fetch-granularity state.

Setting	A100	H100
Sequential process switch within one MIG instance	Prior setting observable	Context setting restored
Concurrent execution across MIG instances	Observable	Observable
Cross-MIG covert channel	Yes	Yes

For processes in different GPU instances, both the A100 and H100 can exhibit cross-instance interference when processes execute concurrently. This occurs because the effect of the L2 fetch-granularity control state can remain visible across GPU instances. Table 1 summarizes these platform-dependent behaviors.

4 RekSai Covert Channel Attack

4.1 Threat Model

We consider multi-tenant GPU-sharing environments in which mutually untrusted users may be co-located on the same physical GPU, including public cloud services, AI clusters, and HPC systems. The attacker consists of two colluding user-level programs: a Trojan (transmitter) and a Spy (receiver), which are common terms used in previous covert channel works [5]. They run as separate OS processes with distinct address spaces and CUDA contexts, and may additionally reside in different MIG instances. We evaluate both software isolation boundaries (separate processes and CUDA contexts) and hardware isolation boundaries (separate MIG instances).

RekSai can transmit data whenever the effect of the L2 fetch-granularity control state is visible across the execution domains of the Trojan and Spy, even though those domains are intended to be isolated by system software. Under this condition, the Trojan can control the microarchitectural state through standard CUDA interfaces, and the Spy can infer the selected state through timing measurements on its own memory accesses. Therefore, the essential requirement is not a particular GPU model, or whether MIG is enabled, but co-residency within a hardware-sharing domain that exposes the same effective granularity behavior.

Both parties require only user privileges, with no administrator access, modified drivers, special runtime support, undocumented interfaces, or physical access. The Spy can allocate GPU memory and read high-resolution GPU timers available to unprivileged CUDA programs. We assume no explicit communication channel, including shared memory or IPC.

We assume only that both parties receive execution opportunities on a shared GPU domain; they need not control SM placement, warp scheduling, cache mapping, or the mechanism that establishes co-location. Once co-resident, RekSai demonstrates an isolation violation wherever the effective granularity state is cross-context visible. In our systems, this condition holds across processes on A100 and across MIG instances on both A100 and H100.

RekSai is a covert channel between two colluding parties rather than a side channel against an honest victim. Our results therefore demonstrate the existence of a non-isolated microarchitectural communication path, not a direct extraction of secrets from an unrelated co-tenant application. The practical risk is that mutually untrusted tenants sharing a GPU may use this channel to bypass intended communication restrictions or policy boundaries, even when no explicit IPC or shared memory is available.

4.2 Attack Workflow

The root cause of RekSai lies in hardware resource sharing despite isolation at the software level. Specifically, when a process reads the L2 fetch granularity, it retrieves the value from its CUDA context. In our experiments, the Spy's `cudaDeviceGetLimit` call always returns its context-local default of 64 bytes, although its L2 timing measurements reflect the value selected by the Trojan. Figure 1 demonstrates the attack workflow for RekSai, comprising a sending phase and a receiving phase.

Sending Phase The Trojan, running with normal user privileges, encodes a data bit by modifying the L2-cache block-fetch granularity inside its own CUDA context. It invokes `cudaDeviceSetLimit` (`cudaLimitMaxL2FetchGranularity`, **var**) to set the granularity parameter *var*, which in turn changes the effective L2 fetch behavior observed by the receiver. Different values of *var* alter the hardware fetch granularity. For example, setting $var = 32$ selects 32-byte fetch units, whereas $var = 128$ selects a 128-byte fetch unit. These architectural changes are visible to other processes on the same GPU and create timing differences that the Spy can later measure. Although the hardware supports three granularity options (32, 64, and 128 bytes), we encode bit-1 using 32 and bit-0 using 128, and avoid using 64. The reason is that 64 bytes is the default CUDA setting, and unrelated applications running on the same GPU may implicitly reset the granularity back to 64. Such unintended resets appear as noise to the Spy and may interfere with decoding.

Receiving Phase To decode the transmitted bit, the Spy selects a 128-byte-aligned memory address whose data is guaranteed to be uncached. On NVIDIA GPUs, each 128-byte L2 cache line is internally divided into four 32-byte sectors, which may be fetched independently. The Spy sequentially accesses four sectors within this 128-byte block and measures the access latency for each. The first sector is always a cache miss and therefore exhibits high latency. However, the timing behavior of the remaining three sectors depends directly on the granularity selected by the Trojan:

- $var = 32$: sectors 2–4 are all cache misses with high latency.
- $var = 128$: sectors 2–4 are all cache hits with low latency, since accessing sector 1 fetches the entire 128-byte cache line.
- $var = 64$: sector 2 is a cache hit, sector 3 is a cache miss, and sector 4 becomes a hit again because accessing sector 3 fetches the corresponding 64-byte block.

By identifying which latency pattern occurs, the Spy can reliably distinguish between bit 1 and bit 0, completing the transmission of a single data bit through the covert channel.

4.3 Constructing the RekSai Covert Channel

We implement RekSai as a practical proof-of-concept (PoC) on real NVIDIA GPUs. At a high level, the Trojan encodes payload bits by selecting an L2 fetch-granularity value, while the Spy decodes them through sector-level timing measurements. Since CUDA contexts are isolated across processes, whereas threads within the same process directly share one context, we focus on the cross-process

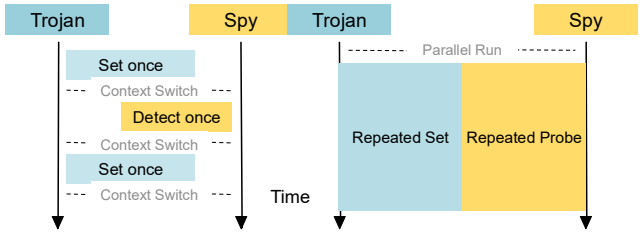


Figure 2: Two execution modes of the RekSai covert channel. Left: in the single-bit mode, the Trojan and the Spy execute in a serialized sender-then-probe manner with explicit context switches. Right: in the multi-bit streaming mode, the Trojan repeatedly updates the L2 fetch granularity while the Spy continuously probes in parallel.

and cross-MIG setting, which captures the actual isolation boundary enforced by CUDA and the operating system.

The covert channel operates in two modes, shown in Figure 2: a *single-bit mode* and a *multi-bit streaming mode*. Both modes rely on the same leakage primitive, but they differ in execution pattern, synchronization requirements, and decoding strategy. The single-bit mode uses serialized turn-taking between the Trojan and the Spy, while the multi-bit mode allows both sides to run continuously in parallel.

Single-bit mode. In the single-bit mode, communication follows a sender-then-probe pattern. The Trojan process encodes one data bit by selecting an L2 fetch-granularity value in its CUDA context and then yielding the GPU. The Spy process subsequently launches a probe kernel and measures the access latency of the four sectors within a 128-byte aligned block. A timing pattern of {Miss, Hit, Hit, Hit} corresponds to bit 0 ($var = 128$), whereas {Miss, Miss, Miss, Miss} corresponds to bit 1 ($var = 32$). Timing is collected using the CUDA `clock()` API with cycle-level precision and does not require elevated privileges.

In our proof-of-concept implementation, the sender first invokes `cudaDeviceSetLimit(cudaLimitMaxL2FetchGranularity, v)` with $v \in \{32, 128\}$ to update the effective L2 cache behavior. The sender then calls `cudaDeviceSynchronize()` before releasing the GPU; the synchronization call establishes a clean experimental handoff rather than selecting the granularity. The receiver then launches one probe kernel to collect the timing. To keep the probed cache line cold, the receiver allocates a 32 MiB device buffer once and shifts the probe start by 4 KiB in each round. To reduce scheduler-induced noise, the timing kernel records measurements only from a fixed SM (SM 0 in our implementation). Overall, this mode provides a clean and controlled demonstration that one granularity update can be reliably observed across process boundaries.

Multi-bit streaming mode. The multi-bit mode extends the same primitive to continuous transmission. The Trojan repeatedly updates the L2 fetch granularity while a long-running Spy kernel continuously scans memory and records sector timings. Compared with the single-bit mode, this design supports streaming transmission but introduces a synchronization challenge.

CUDA provides no intermediate host–kernel synchronization during kernel execution, and the latency of driver-managed granularity updates is undocumented and not directly observable. Consequently, the Spy may oversample the same state or miss transitions; longer update intervals improve stability by allowing the effective behavior to settle before sampling.

To improve decoding robustness, the Trojan inserts the granularity value 64 as an inter-symbol delimiter. These delimiter updates act as anchors in the timing trace, allowing the Spy to distinguish true symbol transitions from repeated observations of the same effective granularity state and to resynchronize after missed updates. In the evaluated implementation, each transmitted symbol is encoded as a two-step sequence: the Trojan first writes the delimiter value 64, and then writes the payload value, using 32 to represent bit 1 and 128 to represent bit 0. On the receiver side, the Spy reconstructs the bitstream offline after the long-running probe kernel finishes and returns the recorded timing trace. When a payload update is observed multiple times, the raw trace contains a run of repeated observations of the same bit value. The decoder collapses such consecutive observations and uses an observed 64 pattern to identify a symbol boundary. If the Spy misses both the delimiter and payload for an entire symbol interval, that symbol may be omitted from the reconstructed stream.

Our evaluated multi-bit prototype uses two MPI ranks with a single `MPI_Barrier` before transmission starts. Rank 0 acts as the Trojan and continuously updates the granularity in a tight loop without introducing sleeps. Rank 1 acts as the Spy, launches one long-running probe kernel in its own CUDA context, and scans a large memory region sequentially. MPI is used only as an experimental convenience: the barrier reduces startup skew, and no MPI communication occurs after the barrier. The channel does not require MPI or a shared Slurm job; it only requires the sender to run within the receiver's probing window.

5 Evaluation

Our experiments are conducted on NVIDIA A100 and H100 GPUs, both of which are widely deployed accelerators for mainstream AI-training and HPC workloads. We evaluate RekSai on A100 under two different environments: a local machine with driver version 470.256.02 and CUDA Toolkit 11.8 and an HPC cluster with driver version 575.57.08 and CUDA Toolkit 12.9. For H100, we evaluate on the HPC cluster with driver version 575.57.08 and CUDA Toolkit 12.9.

5.1 Covert Channel Demonstration

Figure 3 presents the timing patterns observed by the Spy when the Trojan sets different L2-granularity values. The x-axis denotes different sectors across two 128-byte blocks (each sector is 32 bytes), and the y-axis reports the measured access latency in GPU cycles using the `clock()` API. The red horizontal line at 300 cycles separates cache hits from misses for the sectors. In all tested cases, the latency gap between hits and misses is substantial, typically exceeding 200 cycles. For granularity 32, all sectors result in cache misses with high latency because each 32-byte fetch unit is retrieved independently. For granularity 64, accessing sector 1 brings in sector 2, and accessing sector 3 brings in sector 4, yielding a clear

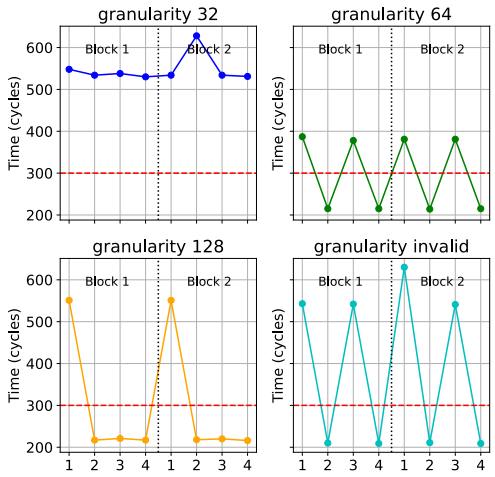


Figure 3: Detected timing patterns for different granularities on the A100. The x-axis represents the 32-byte sectors, and the y-axis shows the measured access latency in cycles. The H100 shows a similar pattern.

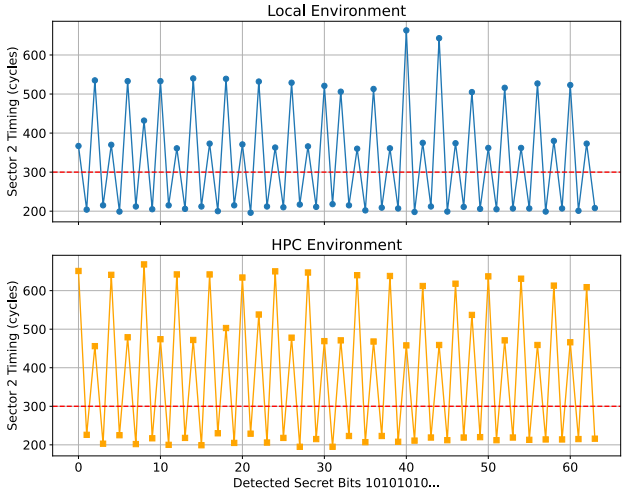


Figure 4: Transferring data bits on A100 in local and HPC environments. For each bit, the Spy accesses sector 1 and measures the timing for sector 2. The miss latency varies with NUMA locality; in this experiment, successive 8 KiB address regions map to different NUMA nodes. The H100 exhibits a similar bit-decoding pattern.

Miss–Hit–Miss–Hit pattern. For granularity 128, accessing the first sector fetches the entire 128-byte line, producing one miss followed by three consecutive hits. We further tested unsupported granularity values (e.g., 256 and -1). Although `cudaDeviceSetLimit` returns success and the sender-side `cudaDeviceGetLimit` reports the attempted value, the measured effective timing pattern for the unsupported values we tested matches the 64-byte behavior.

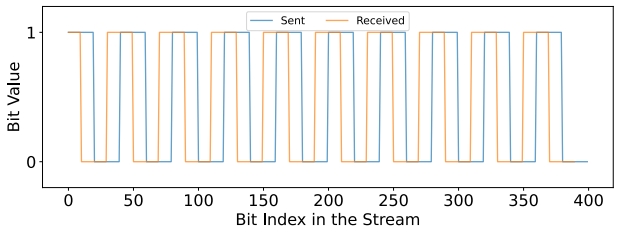


Figure 5: Sent and recovered bitstreams during a long multi-bit transmission on two H100 MIG instances. The A100 exhibits a similar transmission pattern.

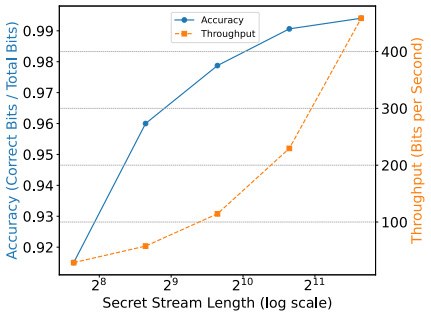


Figure 6: Accuracy and throughput of multi-bit transmission for varying payload lengths on the H100.

5.2 Single-bit Covert Channel Attack

We demonstrate the single-bit version of our covert channel in Figure 4. In this experiment, the Trojan sends a repeating bit stream 101010…, where a bit value of 1 corresponds to granularity 32 and 0 corresponds to granularity 128. The Spy measures the access latency within its own memory space. After accessing sector 1 of a 128-byte block, the Spy records the latency of sector 2, which alone is sufficient to distinguish whether the block was fetched with granularity 32 or 128. For each measurement, the Spy intentionally starts probing from an address offset by 4 KB. This offset is not fixed; any distance is acceptable as long as the target 128-byte block is not initially cached. As shown in Figure 4, the transmitted bits are clearly distinguishable: the latency values fall cleanly below or above the threshold. Interestingly, the cache-miss latencies exhibit a stable two-level structure. **Our reverse-engineering results show that memory allocations can switch NUMA locality at 8 KiB address boundaries, producing distinct memory-access latencies on A100 and H100.** We conduct experiments in a local A100 environment and on an HPC cluster containing both A100 and H100 GPUs, managed by the Slurm job system and running different driver and CUDA versions. Despite these differences, the same timing patterns appear in both environments, with the HPC cluster exhibiting slightly higher miss latencies.

5.3 Cross-MIG Multi-bit Covert Channel Attack

Next, we show that RekSai can violate a stronger isolation boundary, namely MIG isolation. At the same time, we demonstrate a variant of the covert channel that transfers multiple payload bits within a single CUDA kernel execution. For this cross-MIG experiment on the HPC cluster, we submit one two-task Slurm job on a single node. The job script parses `nvidia-smi -L`, selects two MIG UUIDs belonging to the same physical H100 GPU, and launches one MPI rank per MIG instance via `CUDA_VISIBLE_DEVICES`. Concretely, the sender rank is started with the first selected MIG UUID and the receiver rank with the second one, so each process sees only its assigned MIG device while both remain co-located on the same physical GPU. This setup ensures that the Trojan and the Spy run on different MIG instances while still sharing the same underlying physical GPU. Figure 5 shows the transmitted and recovered bitstreams during a long-sequence experiment on an H100 GPU in our HPC cluster. The Trojan and the Spy are placed on two MIG instances of the same GPU, and the assignment is enforced by launching the two MPI ranks with different MIG UUIDs through `CUDA_VISIBLE_DEVICES`. The Trojan continuously transmits 20 consecutive bit-1s followed by 20 consecutive bit-0s in an alternating pattern (blue), while the yellow line shows the bitstream recovered by the Spy. As shown in Figure 5, the recovered sequence closely matches the transmitted one, except for the first few bits, which the Spy misses. We attribute this discrepancy to GPU kernel setup latency. After this warm-up phase, the channel stabilizes and communication proceeds reliably. We further evaluated the cross-MIG channel in 100 trials on each platform, using a 1,000-bit payload in each trial. Across these trials, the normalized bitstream error rate averaged 0.6% on A100 and 0.4% on H100. We also validated the channel without MPI by running the receiver and sender as independent Slurm jobs. The receiver was launched first, followed by the sender with an intentional 100 ms offset. The sender transmitted twenty 64-byte delimiter symbols before a 1,000-bit payload and another twenty after it. In this single validation run, the receiver successfully decoded the entire 1,000-bit payload.

In summary, the cross-MIG experiment shows that the effect of the L2 fetch-granularity control state is visible across the tested MIG instances. This finding has an important security implication: MIG, despite being designed as a stronger isolation mechanism, does not prevent communication through this microarchitectural control-state effect. By demonstrating that the Spy can recover a long transmitted sequence with only minor errors during the initial warm-up phase, we further show that the leakage is stable enough to form a practical covert channel. Thus, RekSai reveals a cross-MIG communication path based on L2 fetch-granularity control state on modern NVIDIA GPUs.

5.4 Throughput and Accuracy Evaluation

In this evaluation, we report the normalized bitstream error rate as the normalized edit distance between the sent and decoded bitstreams, which captures bit flips as well as dropped or misaligned bits. Decoding accuracy is one minus this error rate. We define end-to-end throughput as the number of transmitted bits divided by the total end-to-end transmission time of one run. We first evaluated the throughput of the single-bit version of RekSai on an NVIDIA A100 GPU using the Linux system clock. Because each transmitted bit requires launching a GPU kernel, the total transmission time increases linearly with the payload length. Under this setup, we observed a peak throughput of 6061 bps.

Figure 6 shows the accuracy and throughput of the multi-bit version across different bitstream lengths. Decoding accuracy increases as the stream becomes longer. The initial bit losses are mainly caused by GPU kernel setup latency, and these losses become amortized as the stream grows. For long sequences, the accuracy approaches nearly 99%.

Throughput also increases with stream length. The Spy launches a single long-running kernel that scans a fixed memory region and cannot terminate early. As a result, the total kernel execution time remains almost constant, while a longer payload yields more decoded bits per run. In our experiments, we observed that each transmitted bit corresponds, on average, to approximately 2.3 KB of sequential memory scanning. Under these conditions, the multi-bit version achieved a maximum throughput of 784.6 bps.

This end-to-end throughput includes the receiver's fixed scanning cost, which is amortized as the transmitted payload becomes longer; it therefore underestimates the steady-state bit rate. To characterize steady-state transmission, we transferred 1,000 bits in each of 100 trials and placed timestamps around the steady-state transmission phase. The steady-state bit rate was approximately 0.9 Kbps, and the average normalized bitstream error rate was $0.60\% \pm 0.059\%$ over the 100 trials. This rate is mainly limited by the CUDA runtime calls used by the sender to control L2 fetch granularity.

5.5 Robustness Against Noise

RekSai remains robust under benign co-tenant GPU workloads, but an adversarial co-tenant can disrupt the channel by intentionally injecting the same L2 fetch-granularity symbols used by the sender. We conducted additional cross-MIG experiments to evaluate both cases.

For benign interference, the sender and receiver performed multi-bit transmission on two MIG instances while a third MIG instance ran either (1) a compute-intensive FP32 multiplication kernel or (2) a memory-intensive streaming kernel. The compute-intensive workload did not affect the recovered bits. The memory-intensive workload caused only one additional dropped bit during the initial phase, while steady-state transmission remained robust.

For stronger interference, repeatedly selecting the 64-byte granularity did not significantly degrade the bitstream because the decoder treats 64 bytes as a neutral delimiter rather than a data symbol. In contrast, when the co-tenant repeatedly switched between 32- and 128-byte granularities, the channel was overwhelmed and could not reliably transfer the payload. This is an adversarial setting because the co-tenant actively injects the same symbols used by the sender.

Finally, active and repetitive use of L2 fetch-granularity control appears uncommon in current GPU kernels. In the NVIDIA open-source libraries we inspected, we found no active configuration of this option; the sole occurrence was commented out as an optional performance optimization for filters fitting within a 32-byte sector. Thus, RekSai remains practical under benign co-tenant workloads, while adversarial symbol injection can disrupt the channel.

Table 2: Comparison of RekSai with prior GPU covert channel works. For dependency-related columns (Eviction set, MPS, UVM), R denotes required and NR denotes not required. For the Cross-MIG Attack column, ✓ denotes supported and ✗ denotes not supported.

Work	Main substrate	Eviction set	MPS	UVM	GPU (Released Year)	Cross-MIG Attack
NoC covert channel [1]	On-chip interconnect contention	NR	NR	NR	V100 (2017)	✗
Spy in the GPU-box [5]	Remote GPU L2 cache over NVLink	R	NR	NR	V100 (2017)	✗
(Mis)managed [17]	Shared last-level TLB	R	R (for high-rate channel)	R	1080Ti (2017)	✗
TunneLs for Bootlegging [30]	Shared last-level TLB	R	NR	NR	A30/A100 (2021/2020)	✓
Membar+Load [7]	Cross-partition memory-barrier interference	NR	NR	NR	H100 (2022)	✓
RekSai (ours)	**L2 fetch-granularity control state**	**NR**	**NR**	**NR**	**A100/H100 (2020/2022)**	✓

6 Discussion

6.1 Comparison with Prior GPU Covert Channels

Table 2 compares RekSai with representative prior GPU covert channels. Prior works [1, 5, 17] primarily exploit contention in shared resources, including the on-chip interconnect, remote L2 cache interactions across NVLink-connected GPUs, and the shared last-level TLB. As shown by the dependency columns, these attacks typically require additional conditions, such as eviction-set construction and enabling UVM/MPS.

More specifically, the NoC-based channel [1] relies on contention in the on-chip interconnect. Spy in the GPU-box [5] targets a cross-GPU setting and requires NVLink-connected devices. (Mis)managed [17] leverages shared TLB behavior and depends on UVM and MPS for high-rate transmission. TunneLs for Bootlegging [30] also leverages the last-level TLB, supports communication across MIG instances, and reports higher raw throughput than RekSai.

Compared with these approaches, RekSai's contribution lies in its distinct signaling primitive and setup rather than higher throughput. First, unlike TunneLs, RekSai does not require GPU TLB reverse engineering, TLB set/slice mapping, or eviction-set construction. Second, its sender uses a standard CUDA API to change L2 fetch granularity, and its receiver decodes the signal through L2 sector timing. Third, RekSai highlights limitations in current GPU isolation: our results show that the channel persists across separate processes on A100 and across different MIG instances on both A100 and H100. This observation suggests that software-level isolation abstractions cannot fully isolate all hardware-visible states relevant to covert communication.

Membar+Load, introduced by *Behind Bars* [7], uses memory-barrier-induced interference across MIG L2 cache partitions, where membars issued in one instance affect load timing in another. In contrast, RekSai derives its signal from cross-context visibility of the effective L2 fetch-granularity control state.

The timing primitive may potentially support workload or model fingerprinting; demonstrating such an attack requires additional victim-specific methodology and is left to future work.

6.2 Mitigation

The root cause of RekSai lies in the fact that the effect of the L2 fetch-granularity control state is visible across processes and is not fully isolated during context switches. As a result, once a process updates its granularity setting, the resulting effect can influence the L2 cache behavior observed by other processes. Fully eliminating this vulnerability requires ensuring that this control-state effect does not leak across execution contexts.

A natural but heavyweight mitigation would be to flush or reinitialize the device state at every context switch. In principle, the GPU driver could restore the effective L2 fetch-granularity state upon switching to a different CUDA context, ensuring that architectural settings never cross process boundaries. However, such an approach may introduce nontrivial overhead because context switches occur frequently on modern GPUs, especially under high kernel-launch rates.

From a driver or hypervisor perspective, a software-based mitigation is to prevent unprivileged requests from changing L2 fetch granularity or to isolate their effects by maintaining per-process effective settings. The runtime can enforce independent effective states for each context and block or defer updates that would affect concurrently executing kernels. This approach avoids hardware modifications but increases complexity in the GPU driver stack.

7 Conclusion

This paper presents RekSai, a covert-channel methodology that exploits leakage through GPU LLC control state on Ampere and Hopper architectures. Through reverse engineering of the A100 (Ampere) and H100 (Hopper), we show that the hardware-visible state controlling L2 fetch granularity is not fully confined by intended isolation boundaries. Our experiments demonstrate that this leakage persists across separate processes on A100 and across MIG instances on both A100 and H100. These findings show that CUDA-context isolation and MIG partitioning do not fully isolate this microarchitectural control state, and reveal a covert communication path in current GPU-based multi-tenant environments.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was supported by the National Research Foundation, Singapore (Award NRF2018NCR-NCR002-0001), the Agency for Science, Technology and Research (A*STAR) IAF-ICP (Award I2001E0053), the Singapore Ministry of Education (MOE) (Awards MOE-T2EP20222-0007 and T1 251RES2403), and the China Scholarship Council (Grant No. 202506090026).

References

[1] Jaeguk Ahn, Jiho Kim, Hans Kasan, Zhixian Jin, Leila Delshadtehrani, Wonjun Song, Ajay Joshi, and John Kim. 2021. Network-on-Chip Microarchitecture-Based

Covert Channel in GPUs. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 565–577. doi:10.1145/3466752.3480093

[2] Daniel J. Bernstein. 2005. Cache-Timing Attacks on AES. Technical report. https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[3] Samira Briongos, Pedro Malagón, José M Moya, and Thomas Eisenbarth. 2020. Reload+Refresh: Abusing Cache Replacement Policies to Perform Stealthy Cache Attacks. In *USENIX Security Symposium (USENIX Security)*. 1967–1984. https://www.usenix.org/conference/usenixsecurity20/presentation/briongos

[4] Yujie Cui, Hongwei Cui, and Xu Cheng. 2023. Information leakage attacks exploiting cache replacement in commercial processors. *IEEE Trans. Comput.* 72, 9 (2023), 2536–2547. doi:10.1109/TC.2023.3254918

[5] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2023. Spy in the GPU-box: Covert and Side-Channel Attacks on Multi-GPU Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. 1–13. doi:10.1145/3579371.3589080

[6] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 279–299. doi:10.1007/978-3-319-40667-1_14

[7] Cheng Gu, Reese Levine, Zhenkai Zhang, Tyler Sorensen, and Yanan Guo. 2026. Behind Bars: A Side-Channel Attack on NVIDIA MIG Cache Partitioning Using Memory Barriers. In *Proceedings of the 35th USENIX Security Symposium (USENIX Security)*. USENIX Association, Baltimore, MD, 5373–5392. https://www.usenix.org/conference/usenixsecurity26/presentation/gu-cheng

[8] Yanan Guo, Andrew Zigerelli, Youtao Zhang, and Jun Yang. 2022. Adversarial Prefetch: New Cross-Core Cache Side-Channel Attacks. In *IEEE Symposium on Security and Privacy (S&P)*. 1458–1473. doi:10.1109/SP46214.2022.9833692

[9] Rodrigo Huerta, Mojtaba Abaie Shoushtary, José-Lorenzo Cruz, and Antonio Gonzalez. 2025. Dissecting and Modeling the Architecture of Modern GPU Cores. In *Proceedings of the 58th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 369–384. doi:10.1145/3725843.3756041

[10] Zhe Jia, Marco Maggioni, Jeffrey Smith, and Daniele Paolo Scarpazza. 2019. Dissecting the NVIDIA Turing T4 GPU via Microbenchmarking. *arXiv preprint arXiv:1903.07486* (2019). doi:10.48550/arXiv.1903.07486

[11] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *arXiv preprint arXiv:1804.06826* (2018). doi:10.48550/arXiv.1804.06826

[12] Zhixian Jin, Christopher Rocca, Jiho Kim, Hans Kasan, Minsoo Rhu, Ali Bakhoda, Tor M Aamodt, and John Kim. 2024. Uncovering Real GPU NoC Characteristics: Implications for Interconnect Architecture. In *Proceedings of the 57th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 885–898. doi:10.1109/MICRO61859.2024.00070

[13] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security Symposium (USENIX Security)*. 549–564. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp

[14] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-Level Cache Side-Channel Attacks Are Practical. In *IEEE Symposium on Security and Privacy (S&P)*. 605–622. doi:10.1109/SP.2015.43

[15] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. 2024. Benchmarking and Dissecting the NVIDIA Hopper GPU Architecture. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 656–667. doi:10.1109/IPDPS57955.2024.00064

[16] Xinxin Mei and Xiaowen Chu. 2017. Dissecting GPU Memory Hierarchy through Microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2017), 72–86. doi:10.1109/TPDS.2016.2549523

[17] Ajay Nayak, Pratheek B., Vinod Ganapathy, and Arkaprava Basu. 2021. (Mis)managed: A Novel TLB-Based Covert Channel on GPUs. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. 872–885. doi:10.1145/3433210.3453077

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

[19] NVIDIA. 2023. NVIDIA H100 Tensor Core GPU Architecture. https://resources.nvidia.com/en-us-tensor-core.

[20] NVIDIA Corporation. 2025. NVIDIA CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/.

[21] Amirreza Rastegari, Prabhat Ram, and Michael F Ringenburg. 2025. On the Performance and Scalability of Cloud Supercomputers: Insights from Eagle and Reindeer. In *Proceedings of the SC Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1554–1563. doi:10.1145/3731599.3767705 16th International Workshop on Performance Modeling, Benchmarking, and Simulation of High-Performance Computer Systems (PMBS).

[22] Jeffrey B Rothman and Alan Jay Smith. 2000. Sector Cache Design and Performance. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 124–133. doi:10.1109/MASCOT.2000.876437

[23] Wei Sun, Ang Li, Tong Geng, Sander Stuijk, and Henk Corporaal. 2023. Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2023), 246–261. doi:10.1109/TPDS.2022.3217824

[24] Caroline Trippel, Daniel Lustig, and Margaret Martonosi. 2018. MeltdownPrime and SpectrePrime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols. *arXiv preprint arXiv:1802.03802* (2018). doi:10.48550/arXiv.1802.03802

[25] Matteo Turisini, Mirko Cestari, and Giorgio Amati. 2024. Leonardo: A Pan-European Pre-Exascale Supercomputer for HPC and AI Applications. *Journal of Large-Scale Research Facilities* 9, 1, Article A186 (2024), 16 pages. doi:10.17815/jlsrf-8-186

[26] Wenjie Xiong, Stefan Katzenbeisser, and Jakub Szefer. 2021. Leaking information through cache LRU states in commercial processors and secure caches. *IEEE Trans. Comput.* 70, 4 (2021), 511–523. doi:10.1109/TC.2021.3059531

[27] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Wardlaw Fletcher, Roy H. Campbell, and Josep Torrellas. 2019. Attack Directories, Not Caches: Side-Channel Attacks in a Non-Inclusive World. In *IEEE Symposium on Security and Privacy (S&P)*. 888–904. doi:10.1109/SP.2019.00004

[28] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security Symposium (USENIX Security)*. 719–732. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[29] Yicheng Zhang, Ravan Nazaraliyev, Sankha Baran Dutta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2024. Beyond the Bridge: Contention-Based Covert and Side-Channel Attacks on Multi-GPU Interconnect. In *IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. 35–36. doi:10.1109/SEED61283.2024.00014

[30] Zhenkai Zhang, Tyler Allen, Fan Yao, Xing Gao, and Rong Ge. 2023. TunneLs for Bootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 960–974. doi:10.1145/3576915.3616672

[31] Zhenkai Zhang, Kunbei Cai, Yanan Guo, Fan Yao, and Xing Gao. 2024. Invalidate+Compare: A Timer-Free GPU Cache Attack Primitive. In *USENIX Security Symposium (USENIX Security)*. 2101–2118. https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-zhenkai