

GhostAccess: Attacking the GPU on the Multi-tenant Cloud via CPU LLC under Unified Memory

Zihao Dan^{*†}, Shinan Liu[†], Shuwen Deng[‡], Yinqian Zhang[§], Dongsheng Wang[‡], Yun Chen^{*✉}

^{*}The Hong Kong University of Science and Technology (Guangzhou) [†]The University of Hong Kong

[‡]Tsinghua University [§]Southern University of Science and Technology

danzihao75@gmail.com, shinan6@hku.hk, shuwend@tsinghua.edu.cn, yinqianz@acm.org, wds@tsinghua.edu.cn, yunchen@hkust-gz.edu.cn

Abstract—The emergence of CUDA Unified Memory (UM) provides a unified virtual address space shared by CPUs and GPUs, simplifying programming and enabling memory oversubscription to mitigate GPU memory shortages. However, such technologies may involve a substantial number of CPU-side operations. Whether these behaviors introduce new security risks merits careful investigation.

To answer this question, we reverse-engineered a UM performance API, *cudaMemAdviseSetPreferredLocation*, which enables the GPU to directly access system memory via the CPU page tables by establishing a mapping. Through this reverse engineering, we discover that when the GPU issues write operations to CPU memory through UM mapping, the data is loaded into the CPU last-level cache (LLC), thereby exposing a new attack surface to adversaries. Building on this observation, we propose **GhostAccess**, a cross-tenant GPU attack framework. **GhostAccess** can establish a high-bandwidth channel between a trojan residing on the victim tenant’s GPU and a spy in another tenant, achieving a channel capacity of up to 1.02 Mbps with an error rate of 0.3%, which is $12.7\times$ faster than the state-of-the-art GPU-assisted cross-tenant covert channel. In addition, we design a cross-tenant GPU side-channel attack for **GhostAccess**. We evaluate **GhostAccess** in four scenarios: fingerprinting six machine-learning applications, extracting parameters from K-means and CNN models, and fingerprinting LLM workloads. **GhostAccess** achieves accuracies of 99.6%, 99.8%, 99.6%, and 99.8%, respectively.

Index Terms—side channels, GPU, LLC, cross-tenant attacks.

I. INTRODUCTION

With the emergence of Large Language Models (LLMs), the memory demands of various training and inference workloads have increased dramatically. To address this challenge, CUDA Unified Memory (UM) [42] has become one of the prominent research directions. UM provides a unified virtual address space for the CPU and GPU, allowing both processors to legally access the same pointer. This design greatly simplifies programming. More importantly, UM supports memory oversubscription, enabling users to execute programs whose memory requirement exceeds the physical GPU memory capacity. When GPU memory is exhausted, selected UM pages are evicted to CPU memory. By effectively pooling CPU and

TABLE I: Comparison to Related GPU Side-Channel Attacks.

Work	Bypass MIG	Cross-tenant	Extra HW ¹
GPU TLB [34]	✗	✗	✗
GPU TLB [59]	✓	✗	✗
GPU Cache [9]	✗	✗	✗
GPU Cache [60]	✗	✗	✗
Counters [33]	✗	✗	✗
PCIe (host-GPU) [30]	✓	✗	✗
PCIe (GPU-NIC) [49]	✓	✓	✓
Membars [12]	✓	✗	✗
GhostAccess	✓	✓	✗

¹ In this column, ✓ indicates that the attack requires specialized hardware, such as SmartNIC.

GPU memory, this mechanism significantly mitigates the GPU memory shortage problem.

Although UM offers many advantages, GPU memory accesses via UM may involve a substantial number of CPU-side operations, which raises a concern: **Does CUDA Unified Memory introduce additional security risks to GPUs?**

To answer this question, we conducted a systematic reverse engineering of *cudaMemAdviseSetPreferredLocation* [43], a usage hint API of UM which enables the GPU and CPU to establish a mapping through which the GPU can directly access system memory by using the CPU’s page tables. We found that when the GPU writes data to a system memory address via a UM-established mapping, it affects the state of the CPU LLC. Specifically, *the write operation will force the corresponding cache line to reside in the LLC*.

This implies that UM introduces a new attack surface on the CPU side, potentially enabling GPU attacks in public cloud environments. In practice, cloud service providers normally do not lease the same physical GPU to multiple tenants [1], [5], [7]. Instead, they offer instances that include one or more GPUs alongside a subset of CPU cores. For example, an instance like AWS g6e.xlarge [45] is equipped with one NVIDIA L40S GPU and four vCPUs. These four vCPUs clearly do not constitute all logical cores of a physical CPU, implying that the remaining cores are allocated to other tenants. Compared with technologies such as Multi-Instance GPU (MIG) or vGPU, this provides a stronger level of isolation, since an adversary cannot directly access the GPU allocated to the victim tenant.

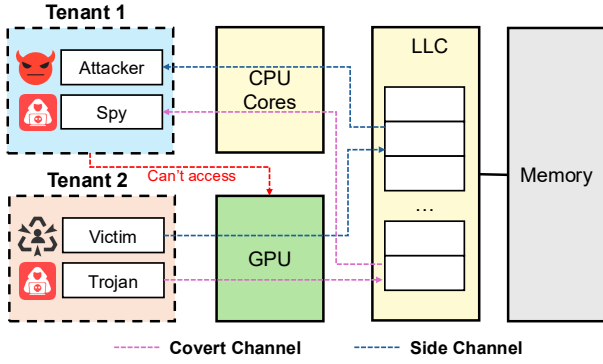


Fig. 1: Overview of GhostAccess.

As a result, most existing GPU side-channel attacks become ineffective under such deployment models.

As summarized in Table I, prior attacks based on GPU TLB [34], [59], cache [9], [60], or performance counters [33] require co-location with the victim (i.e., intra-tenant scenario), which can be readily mitigated by GPU isolation mechanisms such as MIG. Certain PCIe-contention-based attacks (e.g., [30]) are able to bypass MIG by leveraging NVIDIA-provided PCIe monitoring APIs; however, they can only monitor PCIe traffic associated with GPUs accessible to the adversary, and therefore still fail to achieve cross-tenant attacks in public cloud settings. Although [49] demonstrates GPU attacks through PCIe traffic monitoring, it requires access to additional hardware such as SmartNICs within the cloud infrastructure, which is generally impractical in realistic deployment scenarios. [12] found that memory barriers introduced by MIG have exploitable side effects, enabling attacks that bypass MIG isolation. However, their attack still requires the attacker and the victim to share the same physical GPU.

In this work, we design GhostAccess, a cross-tenant GPU attack framework for public cloud environments. The overall attack workflow is illustrated in Figure 1. GhostAccess supports two types of attacks: a covert-channel attack and a side-channel attack.

The covert-channel attack constructs a channel that allows a trojan residing in the victim tenant’s GPU to transmit information to a spy residing in another tenant. Unlike prior GPU-based covert-channel attacks [32], [34], [59], GhostAccess does not require the trojan and the spy to share the same physical GPU in order to communicate. Consequently, NVIDIA’s GPU partitioning mechanism, Multi-Instance GPU (MIG) [35], is ineffective at mitigating GhostAccess. In addition, we propose a concurrent transmission algorithm that fully exploits the massive parallelism of GPUs to realize a covert channel with high bandwidth and a low error rate. Our evaluation shows GhostAccess achieves a channel capacity of up to **1.02 Mbps with an error rate of 0.3%**.

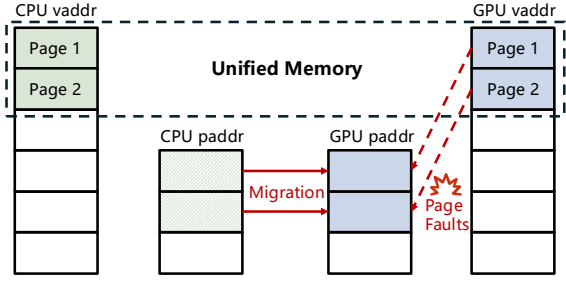
In the side-channel attack setting, we demonstrate a cross-tenant GPU attack in public cloud environments by monitoring the CPU LLC. To the best of our knowledge, this work represents the first fine-grained cross-tenant GPU at-

tack achievable without requiring extra hardware. Moreover, unlike existing side-channel attacks on modern non-inclusive CPUs that target the coherence directory (CD) or snoop filter (SF), GhostAccess directly targets the LLC itself. On non-inclusive CPUs, the LLC is inherently subject to less background noise than the CD/SF-based approaches, which makes GhostAccess **more noise-resilient** than conventional *Prime+Probe* attacks.

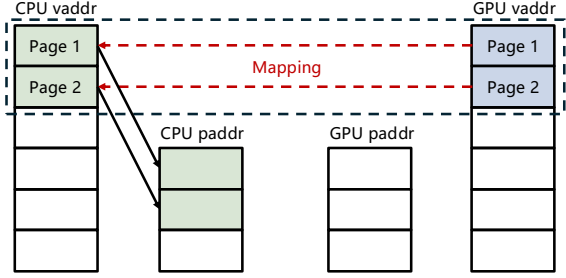
We evaluate the side-channel attacks of GhostAccess under four representative scenarios. In all scenarios, the UM implementations are adopted from UM-accelerated ML libraries [3], [13] or an LLM inference framework [29], which demonstrate that, compared to non-UM implementations, these designs either improve performance or provide memory-oversubscription capabilities. First, in the application fingerprinting setting, we select six widely used machine-learning applications and collect 1000 GPU traces for each application. Using these traces, we train a ResNet-based classifier and **achieve a classification accuracy of 99.6%**. Second, to investigate whether GhostAccess supports finer-grained attacks, we implement a K-means parameter extraction attack. Following a methodology similar to application fingerprinting, **we achieve an attack accuracy of 99.8% across eight parameter configurations**. Next, GhostAccess can also successfully extract the convolution kernel sizes used in CNN models. **The classifier recovers eight different kernel configurations with 99.6% accuracy**, demonstrating that GhostAccess can leak critical architectural information of the victim model. Last, we demonstrate the effectiveness of GhostAccess against memory-intensive workloads such as LLMs. **GhostAccess fingerprints both model architectures and model sizes, achieving an accuracy of 99.8%**.

We argue that the emergence of this vulnerability may stem from GPU vendors lacking full visibility into the effects of UM on the CPU side, as well as insufficient consideration of the associated security risks. The goal of this work is to raise awareness of the new attack surfaces introduced by cross-device memory technologies during system design and to promote the development of a more secure heterogeneous computing ecosystem. The source code of this work can be found at <https://github.com/ZihaoDan/GhostAccess>. In summary, our contributions are as follows:

- We conduct a systematic reverse engineering of the CUDA UM performance-tuning hint `cudaMemAdviseSetPreferredLocation`. Our analysis reveals an undocumented access pattern in the CPU LLC that introduces a previously unexplored attack surface in NVIDIA GPUs.
- We propose GhostAccess, a novel cross-tenant GPU attack framework that exploits this access pattern as the core attack primitive, enabling the violation of tenant isolation on GPUs in the public cloud. To the best of our knowledge, this is the first GPU side-channel attack demonstrated in the practical public cloud.
- We show that GhostAccess achieves a low-error covert channel between the trojan residing in the victim tenant’s GPU and the spy in another tenant. With the proposed con-



(a) GPU behavior when accessing UM residing in CPU memory without using any usage hints.



(b) GPU behavior when accessing UM residing in CPU memory with `cudaMemAdviseSetPreferredLocation` enabled.

Fig. 2: Illustration of CUDA Unified Memory and its Usage Hint `cudaMemAdviseSetPreferredLocation`.

current transmission algorithm, the covert-channel capacity reaches up to 1.02 Mbps with an error rate of 0.3%, which is $12.7\times$ faster than the state-of-the-art GPU-assisted cross-tenant covert channel.

- We implement a cross-tenant side-channel attack on the public cloud. Using this side-channel attack, we successfully fingerprint six machine-learning applications, extract parameters from K-means and CNN models, and fingerprint LLM architectures and model sizes, achieving accuracies of 99.6%, 99.8%, 99.6%, and 99.8%, respectively.

Responsible Disclosure: We responsibly disclosed `GhostAccess` to the NVIDIA Security Team, which has acknowledged and confirmed it as a vulnerability.

II. BACKGROUND

A. CUDA Unified Memory

NVIDIA introduced CUDA Unified Memory (UM) in 2014 to alleviate the growing problem of GPU memory shortage [42]. CUDA UM provides a unified virtual address space that can be accessed by both the CPU and GPU. As shown in Figure 2(a), the physical pages of UM can migrate between CPU and GPU memory. When a processor (CPU or GPU) accesses a UM page that does not reside in its local memory, a page fault is triggered, causing the page to be migrated into the memory of the accessing processor.

CUDA UM offers two main advantages. First, it simplifies programming. With a unified CPU-GPU virtual address space, developers no longer need to allocate separate CPU and GPU memory using `malloc()` and `cudaMalloc()`, nor explicitly migrate data using APIs such as `cudaMemcpy()`. Instead, CUDA

automatically manages data migration between the CPU and GPU upon access to UM, greatly simplifying the programming process.

Second, UM allows programmers to oversubscribe GPU memory. UM allows users to allocate an address space larger than the physical memory of a processor without triggering an out-of-memory fault. When the memory of a processor is exhausted, the CUDA handler will evict certain UM pages and migrate them to another processor to avoid an out-of-memory fault. This technique, which pools system memory and GPU memory together, to some extent alleviates the GPU memory shortage problem.

However, as shown in Figure 2(a), when a processor accesses UM pages that do not reside in its local physical memory, a page fault will be triggered, and the corresponding page will be migrated to the local memory. Therefore, UM also introduces a non-negligible overhead [25]. To mitigate the overhead caused by page faults, NVIDIA introduced prefetching and usage hints in CUDA 8 to help tune performance [43]. NVIDIA provides two main types of performance tuning APIs: **prefetching** and **usage hints**.

This paper primarily focuses on the `cudaMemAdviseSetPreferredLocation` usage hint. As shown in Figure 2(b), its main function is to establish a mapping that allows a processor to access UM data residing in another processor’s physical memory through that processor’s page table, without triggering a page fault. After the programmer asserts `cudaMemAdviseSetPreferredLocation`, the GPU’s first access to the specified address establishes the corresponding mapping. The latency of this access is comparable to, but slightly lower than, that of a page fault, since no data migration is required at this stage. Once the mapping has been established, subsequent accesses to system memory via CPU page-table walks incur significantly lower latency than page faults, although they remain slower than accesses to local GPU memory. It is important to emphasize that the CPU cannot access the GPU’s page table in this manner. This means that the mapping direction can only be from GPU to CPU, or from one GPU to another.

B. Cache Side-Channel Attacks

1) *Contention-based Attacks:* Contention-based attacks are exemplified by *Prime+Probe* [28]. In *Prime+Probe*, the attacker constructs an eviction set that primes the cache set containing the victim’s target address. If the victim accesses that address, some lines from the eviction set are evicted from the cache, which increases the latency of subsequent accesses to the eviction set.

2) *Eviction Set:* In the Prime phase of a *Prime+Probe* attack, the adversary needs to prime an **eviction set** to evict all cache lines currently residing in a target cache set. For a w -way set-associative cache, the adversary must identify at least w addresses that map to the same cache set, referred to as congruent addresses, and group them into an eviction set to ensure that all existing cache lines in the cache set are evicted.

TABLE II: Testbed Configurations

Cloud Provider	CPU	GPU	CUDA Version
Alibaba Cloud	Intel Xeon 8369B	Nvidia A10	12.8
	Intel Xeon 6462C	Nvidia L20	12.8/12.1
	Intel Xeon 8163	Nvidia V100	12.8
AWS	Intel Xeon 8259CL	Nvidia T4	13.0
Mornai	Intel Xeon 8368Q	Nvidia A100	12.4

Therefore, an eviction set must contain at least w congruent addresses.

3) *Existing Methods of Finding the Eviction Set*: The simplest way to construct an eviction set is to **increase the number of attacker-controlled bits**, i.e., by enabling huge pages [20]. However, this approach has two major limitations. First, enabling huge pages typically requires privileged permissions. Second, for the LLC, even if multiple physical addresses share the same LLC set index, they may still be mapped to different slices; as a result, an eviction set cannot be reliably constructed.

Another mainstream approach is to **iteratively search for congruent addresses from a set of candidate addresses** [28]. Suppose the goal is to identify addresses congruent to a target address. Candidate addresses refer to those whose attacker-controlled bits of the cache set index are identical to those of the target address. The basic idea is to repeatedly select addresses from the candidate set and add them to an initially empty eviction set until accesses to the eviction set can evict the target address. At this point, the eviction set contains w congruent addresses as well as multiple unrelated addresses. Then, one address is removed at a time: if the remaining eviction set can still evict the target address, the removed address is deemed unrelated; otherwise, it is identified as a congruent address.

The complexity of this algorithm, as well as the required number of candidate addresses, increases significantly as the number of cache sets and slices grows. Prior work [51] provides a detailed evaluation of its algorithmic complexity and practicality. Moreover, environmental noise can introduce false positives, further increasing the failure probability. To address this issue, [61] proposed an improved algorithm based on binary search and demonstrated its feasibility in public cloud environments.

4) *Our Solution in GhostAccess*: In this paper, we employ the algorithm proposed in [61] to construct all required eviction sets on instances provided by public cloud platforms.

III. REVERSE ENGINEERING

In this section, we discuss the impact on the CPU cache hierarchy when the GPU establishes a mapping to the CPU via the `cudaMemAdviseSetPreferredLocation` and subsequently accesses UM that resides in the system memory through the CPU mapping.

Testbed: all experiments in this section were conducted on five hardware platforms with different CPUs and GPUs from alibaba cloud [5], AWS [1] and mornai [31]. The detail is as Table II. To evaluate the impact of the CUDA version, we

Listing 1 Microbenchmark of GPU Mapping Access

```

1  int main()
2  {
3      //Allocate UM
4      uint64_t *target = nullptr;
5      cudaMallocManaged(&target, size);
6      //CPU initialize the UM
7      CPU_init(target);
8      //Establish the mapping
9      cudaMemAdvise(target, size, flag);
10     //Insert target to specific cache level
11     clflush(target);
12     Fetch(target);
13     //GPU read/write target
14     GPUAccess<<<grid,block>>>(target);
15     cudaDeviceSynchronize();
16     //CPU probe the latency of target
17     uint64_t start = rdtsc();
18     asm volatile ("mfence");
19     _maccess(target);
20     asm volatile ("lfence");
21     uint64_t end = rdtsc();
22     uint64_t latency = end - start;
23
24     return 0;
25 }
```

also conduct experiments with multiple CUDA versions on the same L20 platform.

A. Microbenchmark of GPU Mapping Access

To characterize the impact of GPU accesses to system memory via mapping on the CPU cache hierarchy, we designed a microbenchmark as shown in Listing 1.

The benchmark first initializes the UM on the CPU (line 7) and establishes a GPU mapping via the usage hint `cudaMemAdviseSetPreferredLocation` (line 9), allowing the GPU to access the CPU-side UM through the CPU page table. Then, the target address is flushed from the cache (line 11), and prefetch/demote instructions (e.g., `PREFETCHT0/CLDEMOT0` [19]) are used to place it at a specific cache level (L1/L2/LLC/DRAM). For CPUs predating the Sapphire Rapids architecture, the `CLDEMOT0` instruction is unavailable. Therefore, we use eviction sets targeting the L1/L2 caches to evict the cache line from upper cache levels and place it into the LLC. A GPU kernel is then launched to read or write the target address (line 14). Finally, the CPU measures the access latency to that address (lines 17-22) to determine whether the location of the target address changed after the GPU's access.

To understand the access latency characteristics of different cache levels on the five platforms, we repeatedly measured access latency for various memory locations 10^6 times and recorded the latency distributions, as shown in Figure 3.

B. Characterizing GPU Write via UM Mapping

We placed the target address in different cache locations/-DRAM and executed the microbenchmark with 10^6 GPU write operations. The experimental results are shown in Figure 4. We found that regardless of the initial cache location of the target, after the GPU performed a write operation through UM mapping, the measured CPU access latency on each platform

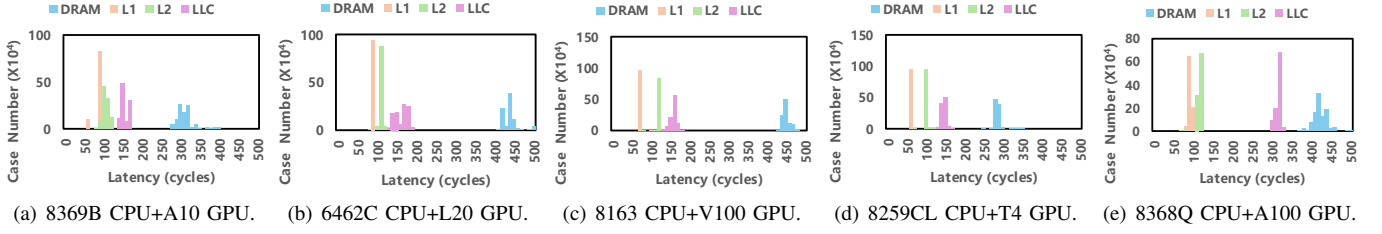


Fig. 3: Access Latency across Different Cache Levels on Five Platforms.

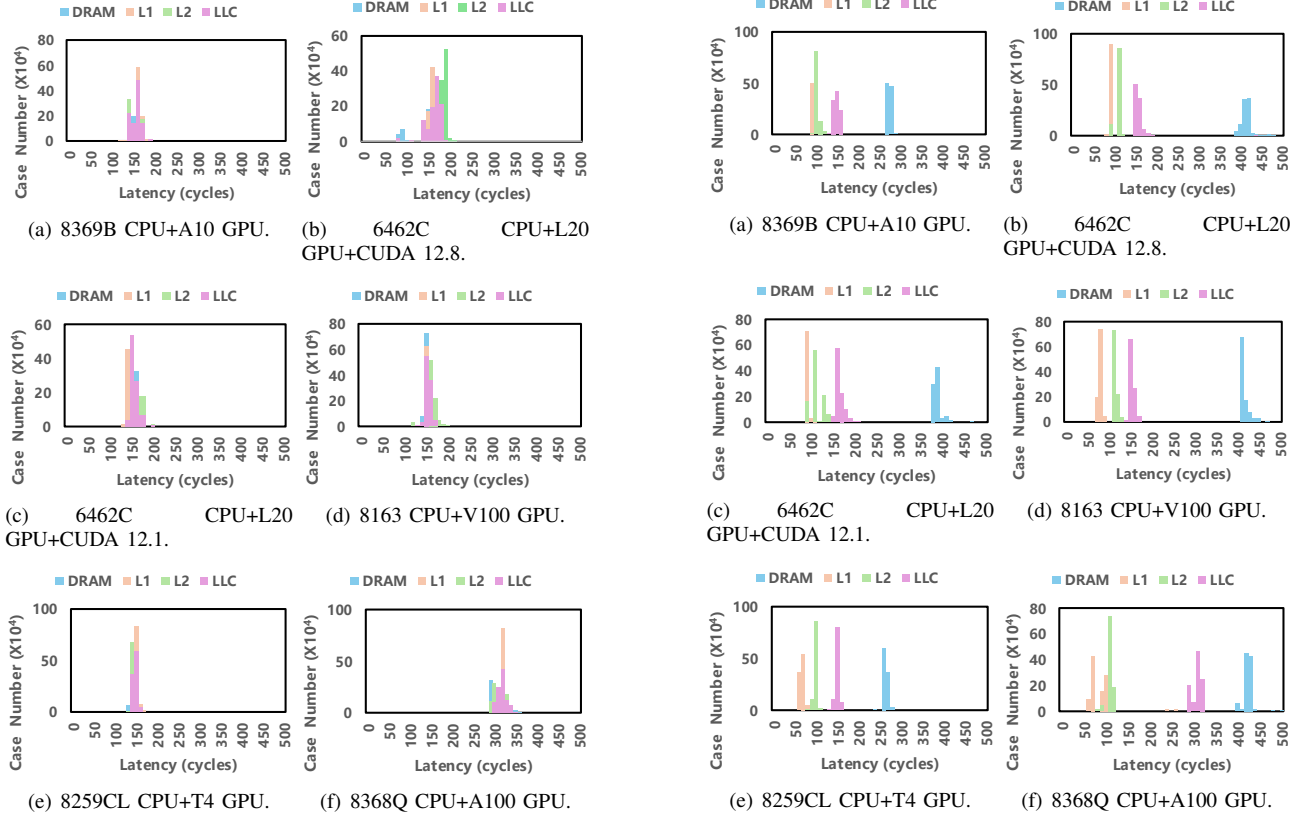


Fig. 4: Access Latency after GPU Write via UM Mapping on Five Platforms.

converged to its corresponding LLC access latency. Therefore, we make the following observation.

Observation 1

After the target address is written by the GPU through UM mapping, it is loaded into the LLC, and all possible cache lines corresponding to this address in the private caches are invalidated.

C. Characterizing GPU Read via UM Mapping

We conducted the same experiment for GPU reads, and the results are shown in Figure 5. Unlike GPU writes, the target's location determines the CPU access latency after the GPU performs a UM mapping read. Moreover, we found that the measured latency matches the original access latency

Fig. 5: Access Latency after GPU Read via UM Mapping on Five Platforms.

of the target's location. Therefore, we make the following observation.

Observation 2

GPU reads through UM mapping do not affect the cache state.

We find that these observations are similar to Intel Data Direct I/O (DDIO) [40]. This suggests that NVIDIA leverages DDIO in its UM mapping implementation. In addition, we find that DDIO provides only marginal performance benefits for UM (see Section VII). Therefore, we hypothesize that NVIDIA adopts DDIO to maintain CPU cache coherence. However, in this paper, we show that such a design introduces a new cross-tenant attack surface, which exhibits even lower noise compared to traditional *Prime+Probe* attacks.

IV. THREAT MODEL

Based on our in-depth characterization of the UM mapping usage hints, we propose *GhostAccess*, a cross-tenant GPU attack framework that includes two powerful attacks: a covert-channel attack and a side-channel attack. Their threat models are illustrated below.

A. Covert-Channel Attacks

For the covert-channel attack in *GhostAccess*, we assume that the trojan resides on the GPU of the victim tenant and writes data to system memory via UM mappings. The spy is another tenant co-located with the victim tenant on the same CPU. As a result, the spy cannot directly access the GPU; instead, it monitors the corresponding CPU cache sets to receive the information transmitted by the trojan. Similar to previous work [2], [15], [28], we assume that the trojan and the spy agree on predefined protocols up front, including synchronization, encoding, and error correction. We further assume that the trojan and the spy have pre-agreed on the target LLC cache sets used by the covert channel. For each cache set, the trojan prepares a UM mapping cache line, while the spy is able to construct the corresponding eviction set. The construction methodology has been described in Section II-B.

B. Side-Channel Attacks

For side-channel attacks in *GhostAccess*, we assume the adversary and the victim are two tenants on the same physical machine. Cloud service providers commonly assign different CPU cores and separate GPUs to different tenants [1], [5], [7], which implies that the adversary cannot access the CPU cores or the GPU used by the victim. Compared to prior GPU side-channel attacks [8], [9], [32]–[34], [59], *GhostAccess* does not require the attacker and the victim to share the same physical GPU, which enables cross-tenant attacks. This also means that our attack is entirely unaffected by NVIDIA Multi-Instance GPU (MIG) [35]. We further assume that the victim performs its computations using UM mapping.

V. COVERT-CHANNEL ATTACKS

In this section, we reveal a CPU–GPU covert channel that leverages the CPU’s LLC. Using this covert channel, a trojan running on a GPU can transmit information to a spy on the CPU even when they do not share the same physical GPU. This scenario commonly arises in multi-tenant cloud environments, where cloud providers allocate multiple cores of the same physical CPU to different tenants, while a GPU is typically assigned exclusively to a single tenant [1], [5], [7].

A. Attack Primitive

GhostAccess implements a covert channel in which a trojan residing on the GPU communicates by writing data to the CPU LLC via UM mappings, thereby evicting a cache line of the corresponding eviction set that has been pre-primed by the spy to convey information. Throughout this paper, we use the term “prime” to denote loading an eviction set into the LLC. Similarly, we use “probe” to denote measuring the

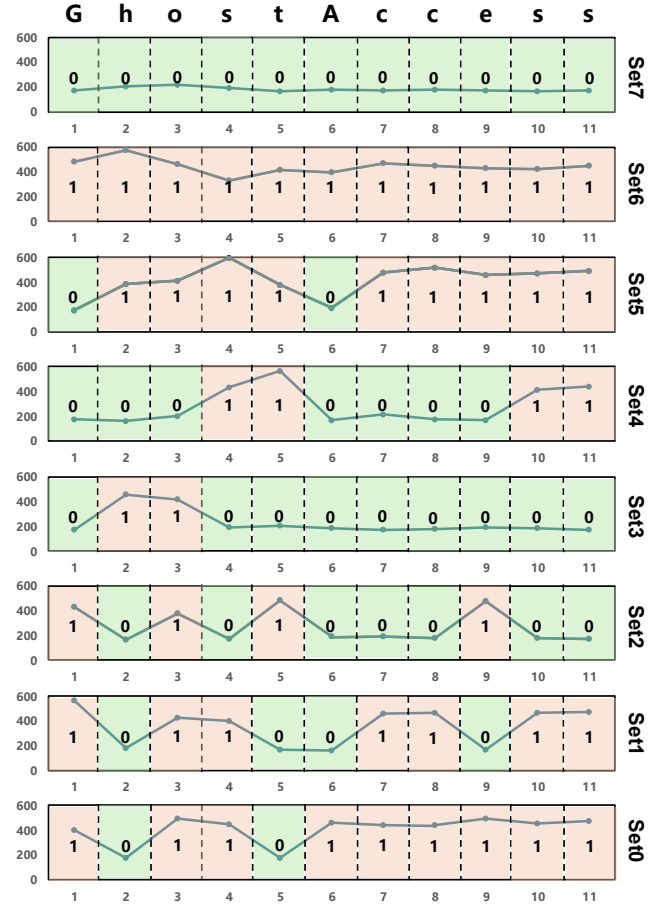


Fig. 6: An 11-Character Concurrent Transmission.

access latency of the eviction set in order to determine whether any cache lines in the eviction set have been evicted from the LLC. The covert channel operates in the following three steps:

- **Step 1:** The spy **prime** the eviction set into the cache set.
- **Step 2:** The trojan encodes the bit via writing the target UM mapping address.
- **Step 3:** The spy **probe** the eviction set to decode the bit.

In *GhostAccess*, when the trojan writes to a UM-mapped address, the spy observes evictions of addresses from the pre-primed eviction set during the probe phase and interprets this event as the transmission of bit ‘1’. Otherwise, if no eviction is observed, the transmission is interpreted as bit ‘0’. Similar to prior work [2], synchronization between the trojan and the spy is achieved using a fixed timing-based delay.

During the probe phase, we employ the parallel probing algorithm proposed in [61] to improve the transmission bandwidth. It leverages the high memory-level parallelism of modern processors and probes all cache lines in the eviction set using overlapped accesses. In our experiments, compared to the conventional probing method, parallel probing reduces the probe latency from 338 cycles to 140 cycles.

B. Concurrent Transmission

GPUs, with their massive parallelism, have become the most widely used processors in scientific computing and machine

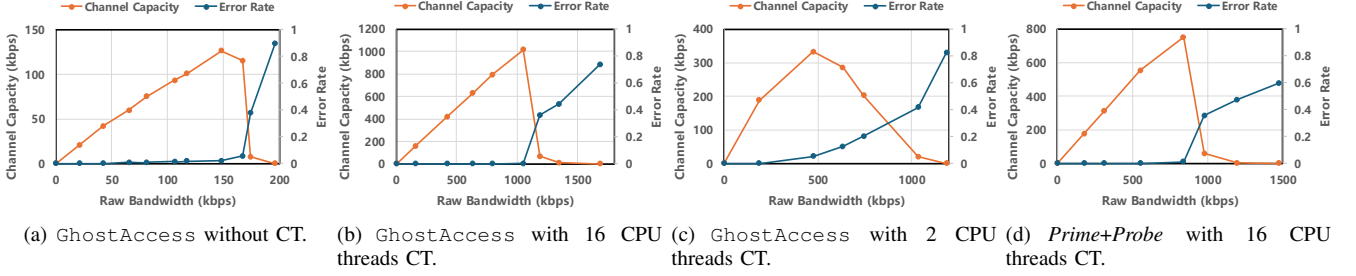


Fig. 7: Capacity and Error Rate of GhostAccess with/without concurrent transmission (CT), and CPU Prime+Probe with CT.

learning. To exploit this parallelism and further increase the bandwidth of the covert channel, we propose a concurrent transmission (CT) algorithm. This algorithm launches multiple threads within a single GPU kernel, each concurrently writing to a target address mapped to a distinct cache set, thereby enabling multi-bit transmission per time window. However, this approach requires priming multiple eviction sets across different cache sets simultaneously. Consequently, on the spy side, multiple CPU cores are needed to perform the priming in parallel. As a result, the degree of concurrency is often bounded by the number of CPU cores available to the spy tenant. Note that, when using n cache sets concurrently for data transmission, the trojan only needs to spawn n GPU threads, whereas the spy typically requires more than n CPU threads, since on non-inclusive CPUs the priming phase additionally relies on helper threads to insert cache lines into the LLC [61].

To ensure practicality, we design an 8-bit concurrent transmission protocol that simultaneously leverages eight cache sets for the covert-channel attack. Such an overhead is reasonable for a typical public-cloud instance equipped with 16–32 vCPUs. With this 8-bit concurrent transmission protocol, each *Prime+Probe* round between the trojan and the spy can transmit one character.

Figure 6 illustrates an example in which 11 characters are transmitted. As shown in the figure, when the trojan does not write data via UM mapping, the spy observes access latencies of the eviction set in the range of 162–222 cycles during the probe phase, with the majority of measurements concentrated around 170 cycles; in this case, the spy interprets the transmitted bit as ‘0’. In contrast, when the trojan writes data through UM mapping, the spy measures eviction-set access latencies ranging from 332 to 600 cycles, with most samples clustered around 470 cycles; this case is interpreted as a transmitted bit ‘1’. According to the predefined transmission protocol, the eight cache sets sampled in each round jointly encode one character. For the first sample, cache sets 7–0 transmit the bit sequence ‘01000111’, which translates to the ASCII character ‘G’. Repeating this process yields a total of 11 transmitted characters, forming the message ‘GhostAccess’.

C. Evaluation

In this section, we evaluate the bandwidth and error rate of the covert channel. To ensure that the two tenants reside on the same physical host, we instantiated two virtual machines (VMs) on a single Alibaba Cloud Elastic Compute Service (ECS) instance equipped with an Intel Xeon Platinum 8369B CPU and an NVIDIA A10 Tensor Core GPU. We allocate 32 vCPUs and an entire GPU to the Trojan VM, while the Spy VM is provisioned with 32 vCPUs but has no access to the GPU.

Similar to prior work [9], [15], [36], [37], [39], to better characterize the capability of the covert channel, we introduce the channel capacity C , defined as shown in Equation 1.

$$C = R \cdot (1 - H(e)) \quad (1)$$

Here, R denotes the raw bandwidth, which is primarily determined by the latency of the prime/probe phases and the transmission interval. The parameter e represents the bit error rate under this raw bandwidth, and $H(\cdot)$ denotes the binary entropy function.

1) *GhostAccess without Concurrent Transmission*: We first evaluate the channel capacity of GhostAccess without applying the concurrent transmission algorithm, i.e., in the setting where each prime/probe round transmits only one bit. The results are shown in Figure 7(a). We gradually shorten the transmission interval to increase the raw bandwidth. Before the raw bandwidth reaches 150 kbps, the transmission error rate remains low; beyond this point, the error rate increases as the raw bandwidth continues to rise. **Ultimately, we observe a maximum channel capacity of 126.2 kbps, achieved at a raw bandwidth of 148 kbps with an average error rate of 2.1%.**

2) *GhostAccess with 16 CPU threads’ Concurrent Transmission*: Next, we evaluate the channel capacity of GhostAccess with the concurrent transmission algorithm enabled, where the spy is allocated 16 logical cores to concurrently launch 8 attacker threads along with their corresponding 8 helper threads. The results are shown in Figure 7(b). Compared to the configuration without concurrent transmission, both the raw bandwidth at which the error rate begins to increase significantly and the maximum achievable channel capacity are substantially improved. **In this setting, we observe**

a peak channel capacity of 1.02 Mbps at a raw bandwidth of 1.05 Mbps, with an average error rate of 0.3%.

3) *GhostAccess with 2 CPU threads' Concurrent Transmission*: To investigate whether the concurrent transmission algorithm itself is effective, rather than the increased transmission bandwidth being solely attributed to allocating more CPU cores, we design the following experiment. We allocate the spy only two CPU cores, identical to the configuration without the concurrent transmission algorithm. Unlike the previous setup, we launch only a single attacker thread and a single helper thread. In each prime phase, we serially prime eight eviction sets, then wait for the GPU to process eight bits simultaneously using the concurrent transmission algorithm, and finally serially probe the eight eviction sets. This design eliminates the performance gains introduced by additional CPU cores, allowing us to isolate the effect of the concurrent transmission algorithm.

We then re-evaluated the channel capacity of *GhostAccess* under this configuration, with the results shown in Figure 7(c). **The results indicate that the maximum channel capacity is achieved when the raw bandwidth reaches 475 kbps, at which point the channel capacity is 332.9 kbps with an average bit error rate of 5.3%.** We observe that the raw bandwidth at which a noticeable error rate begins to emerge is comparable to the case without the concurrent transmission algorithm. Nevertheless, the achieved channel capacity is still significantly higher, demonstrating that GPU-side concurrency makes a concrete and non-negligible contribution to improving the bandwidth of the covert-channel attack.

4) *Prime+Probe with 16 CPU threads' Concurrent Transmission*: Concurrent transmission can also be used to improve the bandwidth of traditional CPU-based Prime+Probe attacks. However, it requires the Trojan VM to occupy another 16 CPU threads concurrently. Compared with the abundant and inexpensive multithreading resources available on GPUs, CPU cores are considerably more valuable and limited.

We also measure the channel capacity of the *Prime+Probe* attack with concurrent transmission on our testbed. The results are shown in Figure 7(d). The maximum channel capacity reaches 749.8 kbps, achieved at a raw bandwidth of 839 kbps with an error rate of 1.4%. Nevertheless, the capacity remains lower than that of *GhostAccess*. We attribute this difference to the inherent synchronization capabilities of GPU threads. As the transmission interval decreases, the error rate of *GhostAccess* increases more gradually than that of *Prime+Probe*, allowing it to sustain higher bandwidth while maintaining reliable communication.

5) *Evaluation on MIG*: To demonstrate that *GhostAccess* remains effective when GPU MIG is enabled and to evaluate the impact of MIG on channel capacity, we partition an NVIDIA A100 GPU into two MIG instances. We then assign separate GPU instances to the Trojan VM and the Spy VM, respectively, and evaluate the channel capacity of *GhostAccess* under this configuration. We also perform a comparison in a non-MIG configuration,

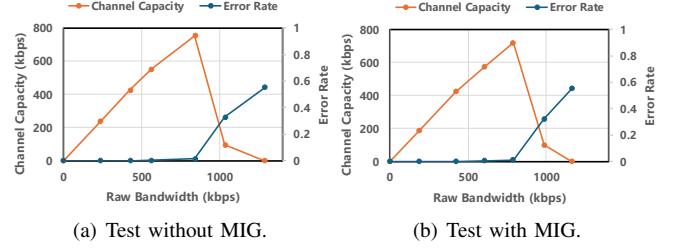


Fig. 8: Capacity and Error Rate of *GhostAccess* on NVIDIA A100 with and without MIG.

where the Trojan VM is assigned an entire GPU while the Spy VM has no access to the GPU. For both settings, we evaluate the channel capacity of *GhostAccess* with 16 CPU threads.

The results are shown in Figure 8. The channel capacity and error rate exhibit the same trends under both configurations. The measured maximum channel capacity with MIG enabled is only 4.7% lower than that without MIG, which we consider to be within normal measurement variation. Overall, MIG has a negligible impact on both the bandwidth and error rate of *GhostAccess*.

D. Comparison with Other Covert-Channel Attacks

Table III summarizes existing covert-channel attacks and compares them with *GhostAccess*. To the best of our knowledge, *GhostAccess* is the first cross-CPU–GPU covert-channel attack demonstrated in a cross-tenant setting. Compared with traditional CPU-based covert channels such as *Flush+Reload* [11] and *Prime+Probe* [57], the parallelism of GPUs enables *GhostAccess* to achieve higher bandwidth. Moreover, conventional CPU covert-channel attacks cannot realize communication across CPU and GPU boundaries.

Although covert channels based on GPU cache contention [9] can achieve high bandwidth, they fail to operate in cross-tenant environments under GPU isolation mechanisms. All covert channels that rely on contention within GPU internal resources, such as GPU TLB-based channels [34], suffer from this limitation. Similarly, covert channels that monitor PCIe traffic via NVIDIA-provided APIs [30] can only observe PCIe activity of GPUs accessible to the attacker, and therefore cannot support cross-tenant attacks. Side et al. [46] demonstrated a covert channel by monitoring data-copy latency from CPU to GPU, which can operate across tenants; however, PCIe contention-based channels generally suffer from low bandwidth, which is 12.7× slower than *GhostAccess*. PCIe-based attacks use contention on the host-GPU PCIe bus as the leakage source. Unlike cache attacks, they do not provide multiple independent components (e.g., LLC sets in *Prime+Probe*) that can be monitored or transmitted over in parallel. Therefore, **concurrent transmission cannot be applied to PCIe-based attacks.**

VI. SIDE-CHANNEL ATTACKS

In this section, we leverage the properties of NVIDIA UM mappings to design a cross-tenant, end-to-end side-channel

TABLE III: Comparison to Related Covert-Channel Attacks.

Work	Cross-Tenant	Capacity	Error Rate
<i>Prime+Probe</i> [57]	✓	200 kbps	- ¹
<i>Flush+Reload</i> [11]	✓	298 kbps	0.01%
GPU Cache [9]	✗	3.55 Mbps	1.3%
GPU TLB [59]	✗	30.79 kbps	0.21%
PCIe Traffic [30]	✗	5.37 kbps	3.3%
PCIe Traffic [46]	✓	80.44 kbps	1.4%
<i>GhostAccess</i>	✓	1.02 Mbps	0.3%

¹ This work does not evaluate the error rate.

attack that targets programs executing on a GPU within the victim tenant.

A. Attack Primitive

In this work, we propose *Prime+Yield+Probe*. The adversary, residing in one tenant, first primes the corresponding eviction set, then yields the CPU core to wait for the victim tenant’s GPU execution, and finally probes the eviction set to determine whether the GPU has accessed the target address. Overall, the *Prime+Yield+Probe* primitive consists of the following three main steps:

- **Step 1:** The adversary **primes** the eviction set into the LLC.
- **Step 2:** The adversary **yields** the CPU core using `sched_yield()` and waits for the victim’s GPU execution to proceed.
- **Step 3:** The adversary **probes** the eviction set to collect the trace.

Test Environments: Most experiments in this section were conducted on an Alibaba Cloud ECS instance, which is equipped with an Intel Xeon Platinum 8369B processor and a single NVIDIA A10 GPU. Following the covert-channel setup, we instantiate two VMs on the same instance: a victim tenant VM with GPU access and 32 vCPUs, and an attacker tenant VM with 32 vCPUs but no GPU access. All traces were collected across all periods of a 24-hour day to ensure that our evaluation captures the worst-case noise conditions observed in public cloud environments.

B. Environmental Noise Evaluation

For modern non-inclusive CPUs, *Prime+Probe* cannot directly target the LLC, because in most cases, memory accesses on non-inclusive architectures only load data into private caches and do not bring it into the LLC. As a result, monitoring the LLC alone is insufficient to observe the execution of victim applications. To address this limitation, existing *Prime+Probe* attacks on non-inclusive CPUs instead monitor the CD or snoop filter (SF). The CD/SF tracks cache lines that reside in private caches in order to maintain coherence when cache lines are modified but not yet written back to memory. Consequently, when the victim loads data into a private cache, the corresponding entry must also be inserted into the CD/SF. This property enables cross-core attacks on non-inclusive CPUs by monitoring the CD/SF.

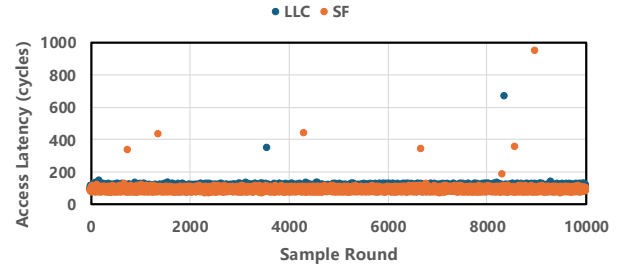


Fig. 9: Results of 10000 *Prime+Probe* Attack Rounds on LLC and SF.

In contrast, when the GPU performs write operations through UM mappings, the target address is directly loaded into the LLC rather than into private caches. As a result, *GhostAccess* targets the LLC itself instead of the CD/SF. Therefore, when monitoring the LLC directly, the attack is subject to significantly less environmental noise compared to attacks that monitor the CD/SF.

To validate our hypothesis, we design the following experiment. For the same set of cache sets, we continuously perform 10000 rounds of *Prime+Probe* on both the corresponding LLC and SF (SF is used in Intel 8369B), in order to observe the impact of environmental noise on their eviction sets. As shown in Figure 9, the eviction-detection threshold is approximately 280 cycles for the LLC eviction set, whereas it is around 110 cycles for the SF. Across 10000 samples, background noise is observed 62 times for the SF, while only 2 such events are observed for the LLC, corresponding to a **96.8% reduction in noise-induced interference**. These results demonstrate that *GhostAccess* is significantly less affected by public-cloud environmental noise than traditional LLC-based side-channel attacks.

C. Handling Numerous Cache Sets

For modern Intel CPUs, the presence of multiple cache slices results in a very large number of LLC cache sets, often on the order of tens of thousands. For example, for the Intel Xeon Platinum 8369B processor used in our evaluation, the LLC contains 65536 cache sets, making it practically infeasible to directly monitor the state of all LLC cache sets. To address this, we aim to identify, prior to launching the attack, the cache sets that are actually exercised during the victim’s execution.

Identifying the cache sets used by the victim’s execution is also a challenge. To locate the target cache sets, we adopt the same approach as prior work [61]: we collect short memory-access traces for candidate cache sets and estimate their Power Spectral Density (PSD) to determine whether they are exercised by the victim. PSD transforms time-domain information into the frequency domain [55]. Unlike background noise, the victim typically performs periodic memory accesses, which manifest as distinct frequency peaks in the PSD of the corresponding memory-access traces.

In the practical attack, we identify only 128 victim-related cache sets as monitoring targets. While the victim may in fact

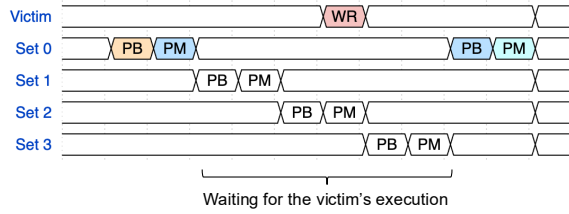


Fig. 10: Attack Workflow of a Single CPU Core in GhostAccess: PM denotes the Prime phase, and PB denotes the Probe phase.

access a larger number of cache sets, we show in subsequent experiments that the information leaked from monitoring just these 128 cache sets is sufficient to enable high-accuracy attacks.

The attacker tenant used in our experiments is equipped with 32 vCPUs, while the attacker needs to monitor 128 LLC cache sets. This implies that each CPU core is responsible for monitoring four cache sets. To this end, GhostAccess adopts the attack workflow illustrated in Figure 10. In the figure, prime and probe phases with the same color correspond to the prime and probe stages of the same *Prime+Probe* round. As shown, the probe phase of the previous round seamlessly overlaps with the prime phase of the subsequent round. However, within a single round, the prime and probe for a given cache set are interleaved with operations on other cache sets. Specifically, after completing the prime on cache set 1, the attacker yields the CPU core to perform probe and prime operations on other cache sets, and probes cache set 1 only when it is scheduled again. The time spent operating on other cache sets thus serves as the waiting period for the victim's execution, which corresponds to the duration of three *Prime+Probe* operations.

D. Application Fingerprinting

In this subsection, we first evaluate the capability of GhostAccess to distinguish different applications. We select six commonly used machine-learning algorithms: Bayesian Network (BN) [18], k-means Clustering (K-means) [16], Convolutional Neural Network (CNN) [26], K-Nearest Neighbors (KNN) [38], Logistic Regression (LR) [23], and Support Vector Machine (SVM) [48]. GhostAccess attempts to differentiate the victim's running application by collecting the access patterns it leaves in the LLC. We use an algorithm implementation from a UM benchmark [13] as a practical victim.

GhostAccess performs 10^5 rounds of *Prime+Probe* attacks on 128 victim-related LLC sets and visualizes the collected traces as memorygrams, which are used as inputs to the classifier. In prior side-channel attack studies, memorygrams have been widely adopted for attacks targeting both CPUs and GPUs. Figure 12 shows example memorygrams for the BN and K-means. Due to space limitations, we only present traces from 10^4 *Prime+Probe* rounds in the figure, while all memorygrams used in the actual evaluation contain access

patterns from 10^5 rounds. We collected 1000 memorygrams for each application at different time periods on a public cloud platform, resulting in a total of 6000 memorygrams used to train a CNN-based classifier. The classifier adopts a ResNet architecture [17]. For each class in the dataset, 700 memorygrams are used for training, 100 for validation, and the remaining 200 for testing.

The classification results on the test set are shown in Figure 11(a). We use a confusion matrix to quantify the classifier's performance. In the matrix, the x-axis denotes the labels predicted by the classifier, while the y-axis denotes the ground-truth labels. The percentage in each cell indicates the probability that samples of a given class are predicted as each label. **The results show that our classifier achieves an accuracy of 99.6%.** Out of 1200 test samples, only five traces are misclassified, all belonging to the K-means, CNN, and LR classes. This further demonstrates that, even when monitoring only 128 victim-related LLC sets, the collected memorygrams still contain sufficient discriminative features to distinguish the victim's running applications accurately.

E. K-means Parameter Extraction

In this subsection, we investigate whether GhostAccess can go beyond application fingerprinting and enable finer-grained attacks, such as inferring specific algorithmic parameters. As a proof of concept, we focus on extracting parameters of the K-means algorithm to demonstrate the fine-grained attack capability of GhostAccess.

K-means is a widely used clustering algorithm based on Euclidean distance [16]. Given a set of samples, K-means will group the samples into k clusters according to pairwise distances, such that points within the same cluster are as close to each other as possible, while the distances between different clusters are maximized.

In the K-means algorithm, the number of clusters k is manually specified by the programmer. In practice, k is typically chosen based on prior knowledge of the data; in the absence of such prior information, an appropriate k can be selected via cross-validation. The choice of k has a significant impact on the behavior and performance of K-means, as it largely determines the quality of the clustering results. This makes the parameter k a valuable attack target. In addition, we argue that the size of the input dataset, i.e., the parameter m , constitutes another valuable target. Accordingly, in this subsection, GhostAccess attempts to recover both the number of clusters k and the input dataset size m by analyzing K-means execution traces under different parameter settings.

Similar to the application fingerprinting attack, GhostAccess performs 10^5 rounds of *Prime+Probe* on victim-related cache sets and collects the resulting traces to construct memorygrams. We evaluate eight different parameter configurations, which are summarized in Table IV. For each configuration, we collect 1000 memorygrams, resulting in a dataset of 8000 memorygrams in total. Among them, 5600 memorygrams are used for training, 800 for validation, and 1600 for testing to train a ResNet-based classifier. The results

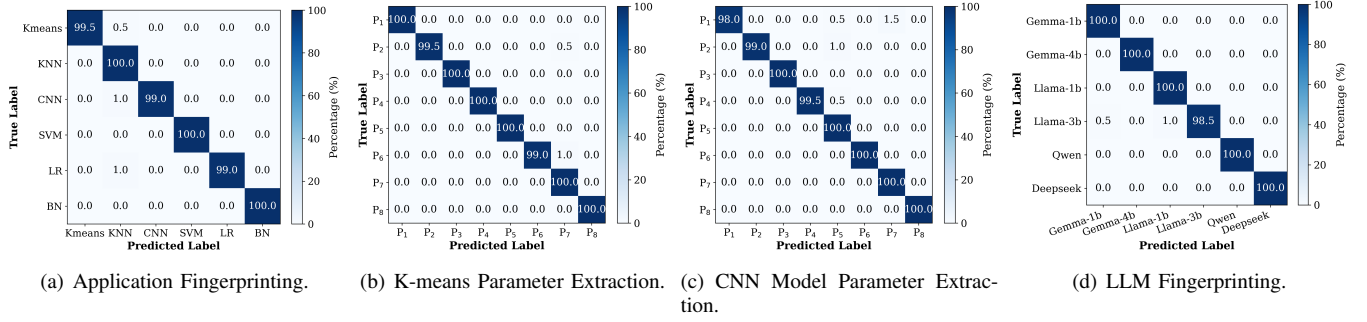


Fig. 11: Confusion Matrix of the Classifier for Four GhostAccess Attacks: (a) 6 Application Fingerprinting Classifier, (b) 8 K-means Configurations Classifier, (c) 8 CNN Kernel Size Classifier, (d) 6 LLM Fingerprinting Classifier.

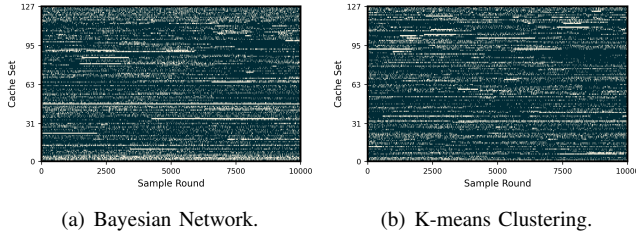


Fig. 12: 10000-Round *Prime+Probe* Attack Traces on Bayesian Network and K-means Clustering.

are shown in Figure 11(b), the overall classification accuracy across the eight parameter configurations reached 99.8%. Among the 1600 memorygrams in the test set, only three samples were misclassified as P_7 : two from configuration P_6 and one from configuration P_2 . These results demonstrate that GhostAccess is capable of extracting key parameters of machine-learning algorithms at a finer granularity while achieving high accuracy.

F. Convolution Layer Parameter Extraction

In this subsection, we further demonstrate that GhostAccess is capable of extracting hyperparameters of a target CNN model. The convolution layer, as a fundamental component of CNN architectures, is responsible for extracting local features from input images. The design of convolution kernels not only determines how features are extracted, but also defines the dimensional relationships of data matrices across different layers. If an attacker can infer the kernel size of each convolution layer, they can approximately reconstruct the architecture of the victim CNN model. For example, distinguishing whether the victim is training LeNet [24] or ResNet [17]. Therefore, in this subsection, we leverage GhostAccess to extract the convolution kernel size, a critical architectural hyperparameter.

In our experiments, we evaluate eight convolution kernel configurations, as summarized in Table IV. Configurations P_1 – P_6 correspond to kernel sizes (kernel height h and kernel width w) commonly used in classical CNN architectures, including ResNet [17], LeNet [24], AlexNet [22], and VGG [47].

In addition, we include two less common configurations, P_7 and P_8 , to extend the evaluation beyond standard settings. Similar to our previous experiments, we collect 1000 memorygrams for each kernel configuration, resulting in 8000 memorygrams in total. Among them, 5600 samples are used for training, 800 samples for validation, and the remaining 1600 samples for testing, in order to evaluate the effectiveness of GhostAccess in extracting CNN hyperparameters. The experimental results are shown in Figure 11(c). **The results show that GhostAccess can extract the convolution kernel size with 99.6% accuracy from another cloud tenant.**

G. LLM Fingerprinting

In this section, we evaluate the effectiveness of GhostAccess against LLM workloads. The attacker has two primary objectives: (1) fingerprinting different model architectures, and (2) inferring the model size within the same architecture family. To demonstrate these capabilities, we use the llama.cpp framework [29] and collect the footprints of different LLM architectures during inference with Unified Memory enabled. Our evaluation covers four LLM architectures: Gemma3 [50], Llama3.2 [10], Qwen3 [58], and DeepSeek-R1 [14]. To further evaluate whether GhostAccess can infer model size, we select two parameter scales from both Gemma3 and Llama3.2. In total, we evaluate six models: Gemma3-1B, Gemma3-4B, Llama3.2-1B, Llama3.2-3B, Qwen3-1.7B, and DeepSeek-R1-Distill-Qwen-1.5B.

Similar to previous experiments, we collect 1,000 memorygrams for each model, resulting in 6,000 memorygrams in total. Among them, 4,200 samples are used for training, 600 samples for validation, and the remaining 1,200 samples for testing. The experimental results are shown in Figure 11(d). **The results show that our attack achieves an accuracy of 99.8%.**

H. Transferability of GhostAccess

1) *Transferability Across Different CUDA Versions:* In Section III, we have already verified that variations in CUDA versions do not affect our observations. In this section, we

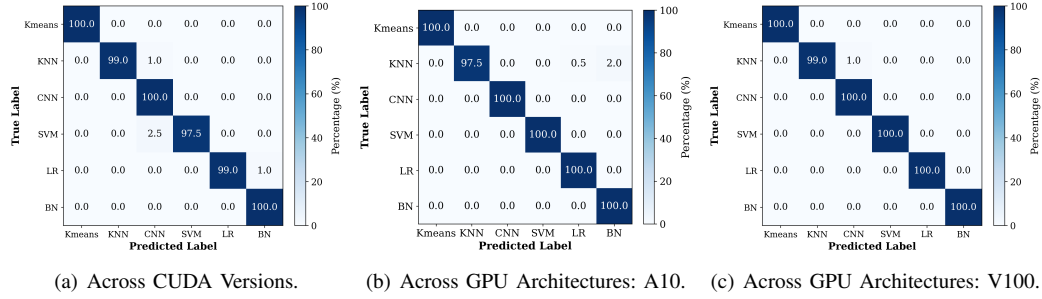


Fig. 13: Transferability Across CUDA Versions and GPU Architectures: (a) Evaluation on CUDA 11.8, (b) Evaluation on NVIDIA A10, (c) Evaluation on NVIDIA V100.

TABLE IV: Configuration of Two Parameter Extraction Attacks.

Victim	Config	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
K-means	Size m	$1e3^1$	$5e3$	$1e4$	$5e4$	$1e5$	$5e5$	$5e5$	$1e6$
	Clusters k	3	3	5	5	9	9	11	11
	Height h	1	2	3	5	7	11	3	7
CNN	Width w	1	2	3	5	7	11	5	6

¹ The notation acb denotes $a \times 10^b$.

further investigate the impact of CUDA version changes on GhostAccess and evaluate its transferability across different CUDA environments.

To this end, we design an experiment to demonstrate GhostAccess is robust to CUDA version differences. Specifically, we collect traces under one CUDA version and use them to train a classifier. We then evaluate the classifier using traces collected under a different CUDA version and measure the resulting classification accuracy.

In this evaluation, we use application fingerprinting as a representative example. For each application, we collect 400 traces under CUDA 12.1 and CUDA 12.8, respectively. From the collected traces of each application, we randomly select 700 traces for training and 100 traces for validation to train the classifier. To evaluate cross-version robustness, we test the trained classifier using 200 traces per class collected under the CUDA 11.8 environment. The resulting confusion matrix is shown in Figure 13(a). The results show that, even without being trained on traces collected from a specific CUDA version, the classifier built by GhostAccess still achieves 99.3% accuracy on that version. This demonstrates the strong transferability of GhostAccess.

2) Transferability Across Different GPU Architectures:

Similarly, in this section, we investigate the transferability of GhostAccess across different GPU architectures, aiming to evaluate its effectiveness when the attacker has no prior knowledge of the victim’s GPU model. To this end, we collect memorygrams from six applications on an Alibaba Cloud V100 ECS instance using the same methodology, and use them as the test set to evaluate the application fingerprinting model trained in our previous experiments. As expected, the model performs poorly in this setting, achieving only 17% accuracy. This is because different GPU architectures induce different access patterns. Consequently, a model trained on

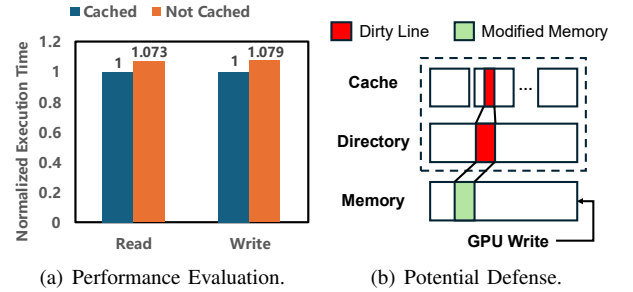


Fig. 14: Evaluation and Protection: (a) Performance Benefits Introduced by CPU LLC and (b) Defense that Mitigates GhostAccess while Preserving Coherence.

A10 memorygrams can hardly recognize the unseen access patterns produced on V100.

This raises a natural question: **if a unified model is exposed to access patterns from multiple GPU architectures during training, can it successfully fingerprint applications across different GPU architectures?** To answer this question, we retrain the model using 4,200 A10 memorygrams and 4,200 V100 memorygrams as the training set, and 600 A10 memorygrams and 600 V100 memorygrams as the validation set. We then evaluate the retrained model using 1,200 A10 memorygrams and 1,200 V100 memorygrams as separate test sets. The results are shown in Figure 13(b) and Figure 13(c). On A10, only five memorygrams from KNN are misclassified, yielding an accuracy of 99.6%. On V100, only two memorygrams are misclassified, achieving an accuracy of 99.8%.

Since the GPU models offered by the cloud provider are public, an attacker can easily include memorygrams from all available GPU architectures during training. This enables the attack to remain effective even when the victim’s GPU model is unknown.

VII. POTENTIAL COUNTERMEASURES

To mitigate GhostAccess, we propose two types of potential countermeasures. The first eliminates, at the root cause, the impact on the CPU LLC when a GPU accesses system memory via UM mapping. The second leverages existing defenses against CPU LLC-based side-channel attacks to mitigate the threats posed by GhostAccess.

The most fundamental mitigation approach is to eliminate the new attack surface introduced by UM design. We compare the performance of UM mapping under two conditions: when the data resides in the CPU cache and when it does not. The results are shown in Figure 14(a). By normalizing the execution time of the cache-resident case to 1, we observe that the performance degradation is minimal in both read and write operations, with execution times remaining around 1000 cycles. This indicates that, for UM, **the performance bottleneck is not primarily determined by whether the data is loaded into the CPU LLC**. Based on this observation, we argue that the security considerations should be prioritized. As illustrated in Figure 14(b), a simple design is to avoid modifying the CPU cache state when the data does not reside in the cache, and directly write it to memory. When the data is cache-resident, it is sufficient to update the directory state to dirty without further perturbing the cache.

Another line of defense is to leverage existing countermeasures against *Prime+Probe* attacks. Over roughly a decade of research, numerous techniques have been proposed to mitigate *Prime+Probe* attacks. One representative approach is to enforce static or dynamic partitioning of the LLC [27], [53]. One defense mechanism is Cache Allocation Technology (CAT) [27], which partitions cache ways among different users to achieve isolation. However, since each user is allocated fewer cache ways, performance degradation is inevitable. Similarly, Randomized Cache designs [41], [56] mitigate side-channel attacks by randomizing cache mappings, but they also incur non-negligible performance overhead. Such trade-offs are often unacceptable for cloud service providers. In addition, determining how to partition cache ways and how to price different cache allocations poses practical challenges for cloud operators. Therefore, we observe that **most cloud computing instances do not deploy these defense mechanisms in practice** [4], [6], [44].

VIII. RELATED WORK

Naghibijouybari et al. [32] constructed covert channels by exploiting GPU warp schedulers, functional units, and memory subsystems. Similarly, Nayak et al. [34] leveraged the TLB to build covert channels within pre-Pascal GPUs, although their attacks cannot bypass the isolation guarantees provided by public cloud. In contrast, Zhang et al. [59] demonstrated that MIG does not partition the last-level TLB and showed a covert channel can be established across GPU instances via the shared last-level TLB. Collectively, these works focus on covert-channel attacks targeting GPUs.

Naghibijouybari et al. [33] leveraged GPU performance counters to perform website fingerprinting, keystroke monitoring, and neural network model recovery; however, the accuracy of their neural network model recovery was limited to 80–87%. Wei et al. [54] extracted DNN hyperparameters across virtual machines by exploiting GPU context-switching penalties, but their attack strictly requires the attacker and the victim to share the same physical GPU. Jiang et al. [21] successfully recovered Advanced Encryption Standard (AES)

keys from GPU cache traces. Dutta et al. [9] demonstrated side-channel attacks across multiple GPUs, but their approach relies on a custom interconnect (NVLink). None of these prior works is able to break the isolation guarantees enforced by the public cloud. Wang et al. [52] extract model layer sequences by exploiting characteristics of UM over PCIe. Similar to [62], this method requires physical access to the server to install specialized hardware, such as a PCIe protocol analyzer.

Purnal et al. [40] reverse-engineered Intel DDIO and leveraged an FPGA to construct eviction sets for attacking CPU programs. In contrast, this paper reveals the previously undocumented use of DDIO in UM designs and focuses on attacking devices (i.e., GPU programs) via DDIO.

IX. CONCLUSION

This paper characterizes the impact on the CPU cache hierarchy when a GPU accesses CPU memory via mappings established by UM. We find that GPU writes through UM mappings load data into the CPU LLC to maintain coherence. Building on this observation, we propose *GhostAccess*, a cross-tenant public cloud attack framework. In *GhostAccess*, we construct a novel covert channel that enables a spy to communicate with a trojan residing in the victim tenant’s GPU. With the proposed concurrent transmission algorithm, the covert-channel capacity of *GhostAccess* reaches up to 1.02 Mbps. Moreover, *GhostAccess* enables cross-tenant side-channel attacks even when the attacker does not share the same physical GPU as the victim. *GhostAccess* achieves over 99% accuracy in fingerprinting six common machine-learning applications, as well as LLM architectures and model sizes, while successfully recovering parameters from K-means and CNN models. Finally, we conclude that fundamentally mitigating *GhostAccess* requires coordinated efforts across vendors to redesign coherence protocols.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their comments. This work was supported by Guangdong Municipal Science and Technology Project (Nos. 2024QN11X196) and Guangdong Basic and Applied Basic Research Foundation (Nos. 2025A1515110151).

REFERENCES

- [1] AWS, “Amazon ec2,” 2026, <https://aws.amazon.com/ec2/>.
- [2] Y. Chen, A. Hajiabadi, L. Pei, and T. E. Carlson, “Prefetchx: Cross-core cache-agnostic prefetcher-based side-channel attacks,” in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 395–408.
- [3] S. Chien, I. Peng, and S. Markidis, “Performance evaluation of advanced features in cuda unified memory,” in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 2019, pp. 50–57.
- [4] A. Cloud, “Instance overview,” 2025, https://www.alibabacloud.com/help/en/ecs/user-guide/overview-52?spm=a2c63.p38356.help-menu-25365.d_0_1_0_0.17e354b4CXij1V.
- [5] —, “Elastic compute service,” 2026, https://www.alibabacloud.com/en/product/ecs?_p_lc=1&spm=a3c0i.7911826.9556232360.77.6a3238708zTIU4.

- [6] G. Cloud, "Machine families resource and comparison guide," 2025, https://docs.cloud.google.com/compute/docs/machine-resource?_gl=1*n070zb*_up*MQ..*_gs*MQ..&gclid=CjwKCAiAqprNBhB6EiwAMe3yhowUkF6bH1RyUv1ka_VsMgUG7rSnhgVZPb2VnMnzpYmNHx1s5IPRoCV8cQAvD_BwE&gclidsrc=aw.ds.
- [7] —, "Google compute engine," 2026, <https://cloud.google.com/products/compute>.
- [8] 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.
- [9] 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.
- [10] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.
- [11] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.
- [12] 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.
- [13] 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.
- [14] D. Guo, D. Yang, H. Zhang, J. Song, P. Wang, Q. Zhu, R. Xu, R. Zhang, S. Ma, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.
- [15] Y. Guo, X. Xin, Y. Zhang, and J. Yang, "Leaky way: A conflict-based cache covert channel bypassing set associativity," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 646–661.
- [16] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.
- [17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [18] D. Heckerman, "A tutorial on learning with bayesian networks," *Learning in graphical models*, pp. 301–354, 1998.
- [19] Intel, "Intel 64 and ia-32 architectures software developer manuals," 2025, <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.
- [20] G. Irazoqui, T. Eisenbarth, and B. Sunar, "SSa: A shared cache attack that works across cores and defies vm sandboxing—and its application to aes," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 591–604.
- [21] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *2016 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 2016, pp. 394–405.
- [22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
- [23] M. P. LaValley, "Logistic regression," *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.
- [24] 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.
- [25] W. Li, G. Jin, X. Cui, and S. See, "An evaluation of unified memory technology on nvidia gpus," in *2015 15th IEEE/ACM international symposium on cluster, cloud and grid computing*. IEEE, 2015, pp. 1092–1098.
- [26] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.
- [27] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *2016 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2016, pp. 406–418.
- [28] 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.
- [29] Llama.cpp, "Llm inference in pure c/c++," 2026, <https://llama-cpp.com/>.
- [30] 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 *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1169–1183.
- [31] Mornai, "Mornai," 2026, <https://mornai.cn/>.
- [32] 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.
- [33] 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.
- [34] A. Nayak, P. B. V. Ganapathy, and A. Basu, "(mis) managed: A novel tlb-based covert channel on gpus," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 872–885.
- [35] Nvidia, "Nvidia multi-instance gpu," 2025, <https://www.nvidia.com/en-us/technologies/multi-instance-gpu/>.
- [36] 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.
- [37] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAM}: Exploiting {DRAM} addressing for {Cross-CPU} attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.
- [38] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
- [39] A. Purnal, F. Turan, and I. Verbaauwhede, "Prime+ scope: Overcoming the observer effect for high-precision cache contention attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2906–2920.
- [40] —, "Double trouble: Combined heterogeneous attacks on Non-Inclusive cache hierarchies," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3647–3664.
- [41] M. K. Qureshi, "Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 775–787.
- [42] N. Sakharnykh, "Beyond gpu memory limits with unified memory on pascal," 2016, <https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal/>.
- [43] —, "Maximizing unified memory performance in cuda," 2017, <https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/>.
- [44] A. W. Services, "Amazon ec2 instance types," 2025, <https://aws.amazon.com/ec2/instance-types/>.
- [45] —, "Amazon ec2 g6e instances," 2026, <https://aws.amazon.com/ec2/instance-types/g6e>.
- [46] 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.
- [47] K. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [48] S. Suthaharan, "Support vector machine," in *Machine learning models and algorithms for big data classification: thinking with examples for effective learning*. Springer, 2016, pp. 207–235.
- [49] M. Tan, J. Wan, Z. Zhou, and Z. Li, "Invisible probe: Timing attacks with pcie congestion side-channel," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 322–338.
- [50] G. Team, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

- [51] P. Vila, B. Köpf, and J. F. Morales, “Theory and practice of finding eviction sets,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 39–54.
- [52] Z. Wang, X. Zeng, X. Tang, D. Zhang, X. Hu, and Y. Hu, “Demystifying arch-hints for model extraction: An attack in unified memory system,” *arXiv preprint arXiv:2208.13720*, 2022.
- [53] Z. Wang and R. B. Lee, “New cache designs for thwarting software cache-based side channel attacks,” in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 494–505.
- [54] 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.
- [55] P. Welch, “The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms,” *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 2003.
- [56] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, “{ScatterCache}: thwarting cache attacks via cache set randomization,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 675–692.
- [57] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, “Attack directories, not caches: Side channel attacks in a non-inclusive world,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 888–904.
- [58] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, “Qwen3 technical report,” *arXiv preprint arXiv:2505.09388*, 2025.
- [59] Z. Zhang, T. Allen, F. Yao, X. Gao, and R. Ge, “T unne l s for b ootlegging: 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*, 2023, pp. 960–974.
- [60] 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.
- [61] Z. N. Zhao, A. Morrison, C. W. Fletcher, and J. Torrellas, “Last-level cache side-channel attacks are feasible in the modern public cloud,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 582–600.
- [62] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, “Hermes attack: Steal {DNN} models with lossless inference accuracy,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.