NVBleed: Covert and Side-Channel Attacks on NVIDIA Multi-GPU Interconnect

Yicheng Zhang
University of California, Riverside
Riverside, CA, USA
yzhan846@ucr.edu

Ravan Nazaraliyev
University of California, Riverside
Riverside, CA, USA
rnaza005@ucr.edu

Sankha Baran Dutta
Brookhaven National Laboratory
Upton, NY, USA
sdutta2@bnl.gov

Andres Marquez
Pacific Northwest National Laboratory
Richland, WA, USA
Andres.Marquez@pnnl.gov

Kevin J. Barker
Pacific Northwest National Laboratory
Richland, WA, USA
kevin.barker@pnnl.gov

Nael Abu-Ghazaleh
University of California, Riverside
Riverside, CA, USA
naelag@ucr.edu

Abstract

Multi-GPU systems are becoming increasingly important in high-performance computing (HPC) and cloud infrastructure, providing acceleration for data-intensive applications, including machine learning workloads. These systems consist of multiple GPUs interconnected through high-speed networking links such as NVIDIA's NVLink. In this work, we explore whether the interconnect on such systems presents a source of leakage, enabling new forms of covert and side-channel attacks. Specifically, we reverse-engineer the operations of NVLink-V1, V2, and V3 and identify two primary leakage sources: timing variations caused by contention and performance counters that reveal communication patterns. The leakage sources are visible from a remote GPU, enabling potentially dangerous attacks. We further validate their presence across three generations of NVLink, as well as on AMD and NVIDIA multi-GPU configurations connected via PCIe. Building on these observations, we develop contention-based covert-channel attacks across two GPUs, achieving a bandwidth of over 70 kbps with an error rate of 4.78% for the contention channel. We also develop two end-to-end side-channel attacks: application fingerprinting (including 18 high-performance computing and deep learning applications) and 3D graphics character identification within *Blender*, a multi-GPU rendering application. These attacks are highly effective, achieving F1 scores of up to 96.77% and 86.86%, respectively. We also discover that leakage surprisingly occurs across virtual machines on the Google Cloud Platform (GCP) and demonstrate a side-channel attack on Blender, achieving F1 scores exceeding 88%. We also explore defenses against cross-VM attacks by adding differential privacy noise to obscure side-channel leakage.

CCS Concepts

• **Security and privacy → Side-channel analysis and countermeasures**.

CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832603

Keywords

Side-channel attacks, covert channels, GPU security, multi-GPU interconnects, NVLink, cross-VM attacks

ACM Reference Format:
Yicheng Zhang, Ravan Nazaraliyev, Sankha Baran Dutta, Andres Marquez, Kevin J. Barker, and Nael Abu-Ghazaleh. 2026. NVBleed: Covert and Side-Channel Attacks on NVIDIA Multi-GPU Interconnect. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3832603

1 Introduction

Graphics Processing Units (GPUs) have emerged as a primary platform for supporting data-intensive applications. These applications range across a number of domains, including machine learning and natural language processing, scientific simulations, cryptocurrency mining, and 3D graphics rendering. As the size of these problems continues to increase, multi-GPU computing is needed to match this expanding demand, which far surpasses the computational and memory resources of a single GPU. For example, training large language models (LLMs) is only possible through large numbers of GPUs: the LLaMA 65B-parameter model leveraged 2048 NVIDIA A100 GPUs for 21 days of training [71]. Similarly, Smith et al. [63] trained the Megatron-Turing Natural Language Generation model (MT-NLG) on NVIDIA's Selene supercomputer, utilizing 560 DGX A100 nodes (several thousand GPUs). Despite the widespread adoption of GPUs on leading cloud platforms such as Amazon Elastic Compute Cloud (EC2), Microsoft Azure, and Google Cloud Platform (GCP), where they often process sensitive private information, research into their security remains in its early stages.

This paper proposes NVBleed, a side-channel attack targeting the NVLink interconnect in multi-GPU systems. Server-class NVIDIA GPUs leverage high-bandwidth interconnects such as NVLink and NVSwitch to achieve high throughput and low-latency communication. These links are used to communicate between the GPUs, either explicitly using commands that copy data from one GPU to another or implicitly through transactions that access shared memory that is mapped to remote GPUs. These communication patterns vary across applications and workloads; if an attacker can observe them, they may infer information about the victim application.

Several prior papers [1, 23, 41, 42, 45, 60, 75, 78, 80, 81] have studied covert- and side-channel vulnerabilities on GPUs, primarily focusing on single-GPU resources or CPU-GPU interactions [15]. These attacks generally require a spy process to be co-located with the victim on the same GPU or within the same integrated CPU-GPU system. One prior work attacked multi-GPU settings by exploiting cache contention to infer memory access patterns [16]. NVBleed also belongs to the broader category of side-channel attacks on interconnect, but it differs in important ways. Prior interconnect attacks mainly target fabrics that serve cache accesses, memory transactions, or CPU-peripheral I/O activity. In contrast, NVBleed exposes GPU-to-GPU communication behavior over discrete multi-GPU interconnects, including NVLink and PCIe GPU-to-GPU paths. This leakage expands the attacker model beyond same-GPU co-location and, in some settings, enables leakage across isolated VM instances. Moreover, NVLink is a distinct interconnect whose internal behavior must be reverse-engineered to develop effective leakage measurement and correlation mechanisms. To the best of our knowledge, this is the first work to exploit multi-GPU interconnect communication, including NVLink and PCIe GPU-to-GPU paths, as a source of side- and covert-channel leakage.

We reverse-engineer NVLink (V1–V3) to characterize leakage sources exploitable by attackers. We observe timing differences in the NVLink or PCIe transactions that leak information about other ongoing communication. Moreover, NVIDIA provides performance counters related to NVLink transactions that leak information across applications and even across instances in the real cloud. Through monitoring these leakage sources, we reverse-engineer the NVLink packet format and communication patterns between multiple GPUs (discussed in § 3), and leverage this information to build covert and side-channel attacks.

We demonstrate our attacks on five multi-GPU platforms, including a DGX-1 system with NVLink-V1, GCP servers equipped with NVLink-V2, NVLink-V3, and PCIe Gen3, and an AMD server with PCIe Gen4, showing that the observed interconnect side channels are not limited to a single GPU generation or platform. First, we develop contention-based covert-channel attacks (*ContenLink*) in which a sender program on one GPU transmits information covertly to a receiver program on another GPU (discussed in § 4). We optimize the channel by incorporating additional parallelism, attaining a bandwidth of 70.59 kbps, with an error rate of 4.78% for the contention channel. We also carry out two end-to-end contention-based side-channel attacks. In the first attack, we use the observed leakage to fingerprint applications such as deep learning models and molecular dynamics simulation benchmarks (discussed in § 4.2.1), with an F1 score of 96.77%. The second attack is a proof-of-concept attack on the popular Blender rendering toolkit [11] (discussed in § 4.2.2). This attack shows that workload-dependent NVLink traffic patterns can reveal coarse-grained rendering properties. In this controlled setting, the attacker can identify which 3D graphics character is being rendered by a victim user, with an F1 score exceeding 86%.

While exploring this attack on the cloud, we discovered, surprisingly, that there is measurable leakage across VM instances that use different GPUs and that do not communicate. We leverage this leakage to develop a cross-VM side-channel attack on the public cloud platform GCP, achieving an F1 score of over 88% in correctly identifying 3D rendered characters (discussed in § 5). Finally, we discuss the challenges associated with mitigating these attacks and propose four potential solutions (*e.g.*, adding Differential Privacy (DP) noise to obscure the side channel) in § 6.

In summary, the contributions of this paper are:

- We reverse-engineer the NVLink interconnect (V1/V2/V3) and identify two leakage sources that disclose details of its operation.
- We design contention-based covert-channel attacks and two side-channel attacks: (1) application fingerprinting and (2) identification of 3D graphics character rendering on victim GPUs.
- We demonstrate NVLink leakage across co-located VM instances and develop a cross-VM side-channel attack that identifies 3D graphics characters.
- We discuss and evaluate possible mitigations, including adding Differential Privacy (DP) noise to obscure the side channel in cross-VM attacks.

Disclosures and ethics. All experiments were conducted safely on dedicated lab testbeds or isolated cloud sandboxes, with no impact on other users. We disclosed our findings to NVIDIA and Google. Google has reviewed our report, filed an internal bug, and indicated that the product team will evaluate whether mitigation or fixes are necessary.

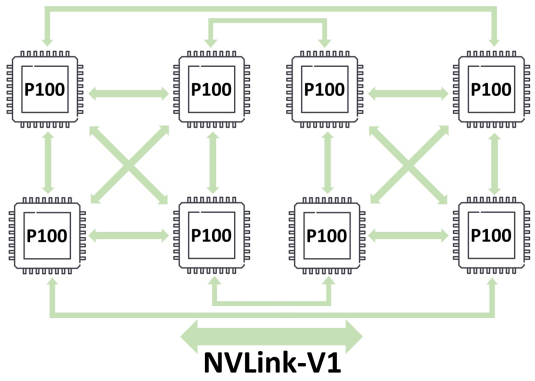
2 Background and Threat Model

In this section, we provide background on multi-GPU interconnects (*e.g.*, NVLink and PCIe) and an overview of the GPU performance monitoring unit. We then present our threat model for two attack setups, summarize the attacker's capabilities in each, and review the relevant attacks.

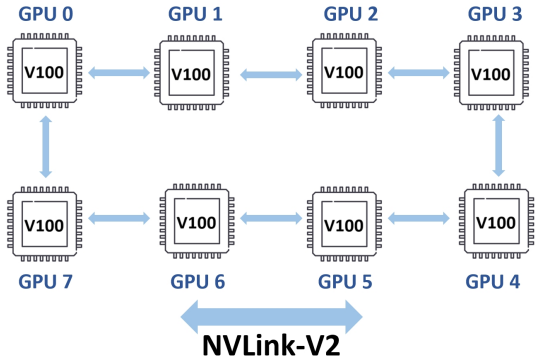
2.1 NVLink and PCIe

NVLink is a high-speed, high-bandwidth interconnect technology developed by NVIDIA [48] to support data sharing among peer GPUs [37], as well as between CPUs and GPUs. In some configurations, there are multiple NVLink physical connections between neighboring GPUs. Each NVLink connection supports communication in both directions. In this paper, we use *slot* to refer to one such physical NVLink connection. NVLink supports communication between GPUs in three main ways: (1) explicit peer-to-peer (P2P) communication, where GPUs copy user-specified buffers through `cudaMemcpyPeer()`; (2) peer access to device memory allocated via `cudaMalloc()`, where memory allocated on one GPU can be accessed by another GPU without staging through host memory; and (3) unified virtual memory, where GPUs can access memory mapped on remote GPUs. In unified virtual memory, data may be migrated at page granularity or accessed at cache-line granularity, depending on programmer hints and runtime decisions.

NVLink has evolved through several generations, including V1, V2, V3, and NVSwitch-based systems. NVLink-V1, introduced with NVIDIA's Pascal P100 GPU [13], provides four NVLink slots per GPU, each delivering 20 GB/s. NVLink-V2, released with the Volta V100 [10], increases this to six slots at 25 GB/s each. Fig. 1 illustrates the GPU topologies of these generations. NVLink-V3 and NVSwitch, deployed with the A100 GPU [9], further expand connectivity: A100 GPUs support 12 NVLink slots at 50 GB/s each, while



(a) DGX-1 system – P100 GPUs with 4 NVLinks.



(b) GCP 8-GPU machine – V100 GPUs with 6 NVLinks.

Figure 1: GPU topology of two experimental machines.

NVSwitch provides high-bandwidth, all-to-all GPU connectivity within a server. Recently, a consortium including AMD and Intel introduced Ultra Accelerator Link (UALink) as an open accelerator interconnect standard [18]. In this study, we evaluate all three NVLink generations, V1–V3.

Peripheral Component Interconnect Express (PCIe) is the standard interface for connecting high-speed I/O devices [6]. It employs point-to-point serial links and has become the de facto choice for peripherals such as CPUs, GPUs, and NVMe SSDs, and is widely deployed in both personal computers and servers. A PCIe link may consist of 1, 2, 4, 8, or 16 lanes, with bandwidth determined by both the number of lanes and the generation of the standard. For instance, PCIe Gen3 provides approximately 1 GB/s per lane per direction, yielding approximately 16 GB/s per direction, or 32 GB/s of aggregate bidirectional bandwidth, on a ×16 link. In this work, we evaluate contention-based timing leakages in PCIe connections on both AMD and NVIDIA multi-GPU platforms (e.g., AMD Radeon Instinct MI50 and NVIDIA Tesla P100).

2.2 GPU Performance Counters

GPU vendors have introduced performance monitoring units, similar to those available on CPUs, to help developers understand and optimize application performance. While these tools provide valuable insights, previous works [3, 31, 42, 55, 60, 68, 74, 75, 77] have shown they can be sources of side-channel leakage. The NVIDIA GPU performance counters are accessed through the CUDA Profiling Tools Interface (CUPTI) [50]. There are a number of NVLink-related performance counters; for instance, *nvlink_total_data_received* tracks the number of bytes of data received across all NVLinks on a particular GPU. AMD also offers analogous APIs for profiling GPU memory and communication transaction metrics [4]. We comprehensively overview all available NVLink-related performance counters in Section 3.2.

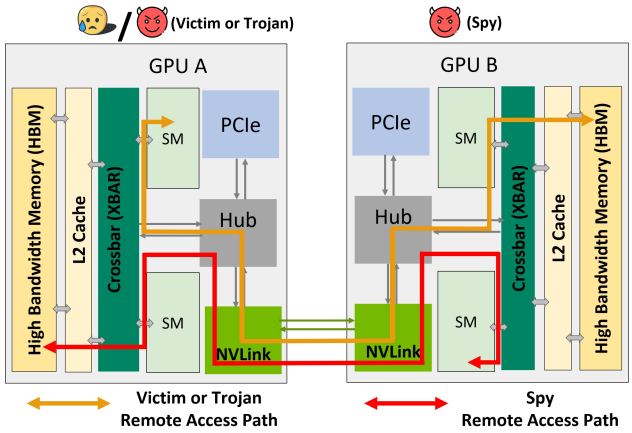


Figure 2: Covert and side-channel attacks in a GPU box.

2.3 Threat Model

We consider two attack setups: (1) Attacks in a GPU box, where multiple applications are co-located on the same multi-GPU system and share NVLink connections. Essentially, these are servers with multiple interconnected GPUs that host applications from multiple users, similar to the threat model assumed by Dutta et al. [16]. This setup is common in DGX systems such as DGX-1 (Pascal and Volta), DGX-2, and DGX A100, and is widely deployed in large research labs where we conducted some experiments. In this environment, an application can map pages on remote GPUs and initiate transfer transactions without requiring a kernel to execute on the remote GPU; (2) Attacks in the cloud, where multiple VMs are assigned GPU(s) on the same physical host via passthrough mode. While each VM controls its own GPU(s), some NVLink slots connecting GPUs on the same machine but that belong to different VMs remain unused. For example, Google Cloud Platform offers nodes with eight GPUs connected in a 1-D NVLink torus, where multiple VM instances can each own a subset of GPUs. In such cases, the physical links between GPUs belonging to different instances remain unused.

Setup 1: Attacks in a GPU box. In this scenario, the attacker and victim reside on two separate GPUs connected by the same NVLink. Fig. 2 illustrates this setup. The attacker can leverage the following capability: **Measuring NVLink timing contention**. Without superuser privileges, the attacker can create contention on the shared interconnects (*e.g.*, NVLink or PCIe) and measure timing variations to infer the victim's activity.

Setup 2: Attacks in the cloud. In this scenario, the attacker and victim run on different VM instances, but their GPUs share an

NVLink connection (see Fig. 12). Because inter-instance links remain unused, the attacker cannot create direct contention, and timing-based leakage is not observable. However, we found that leakage still occurs through performance counters, even on inactive links. In this setup, the attacker can **access NVLink performance counters**. NVIDIA provides counters that report traffic information on shared NVLink connections. This leakage source is often disabled: NVIDIA released a driver patch [51] that restricts user-mode access in response to side-channel attacks that demonstrated that they can be exploited [42]. As a result, this vector is exploitable only if the patch is not applied (e.g., when counters remain enabled for autotuning optimizations [59]) or if the attacker can request downgraded drivers to restore access.

Performance counter access through GPU driver downgrade for Setup 2. There are legitimate uses for access to performance counters, and although they are disabled for security, major cloud providers (e.g., AWS, GCP, Azure, and Alibaba) allow users to enable them or install any version of GPU drivers in their own instance. We confirmed that on AWS, GCP, and Azure, a user can downgrade the GPU driver (for example, Tesla P100, V100, and A100 GPUs) inside their own VMs to a version predating the driver patch [51]. Note that we do not assume the victim's drivers are modified, only that the attacker can downgrade the drivers in their own instance.

3 Demystifying NVLinks

In this section, we first reverse-engineer the NVLink operation. Next, we analyze the NVLink-related performance counters and identify their leakage. Finally, we study the timing behavior when NVLink or PCIe contention arises to identify timing-related leakage.

Experiment platform. We evaluate our work on five testbeds, including a local DGX-1 system and several Google Cloud servers (details in Table 1), covering NVLink-V1 to V3 as well as PCIe Gen3 and Gen4. Fig. 1 illustrates the NVLink topologies of these systems: the DGX-1 employs a hypercube topology, while the GCP 8-GPU platform adopts a ring topology. We tested the GCP VMs in three different regions: us-central1-c, us-east1-c, and us-west1-a.

3.1 Reverse Engineering NVLink Operation

NVLink operates as a packet-based interface, where each packet can contain multiple Flow Control Units (flits). Fig. 3 illustrates the format of an NVLink packet. In this section, we use NVLink-related counters to reverse-engineer NVLink characteristics, such as the packet format.

NVLink-V1 (DGX-1 system). As discussed in an NVIDIA whitepaper [22], NVLink-V1 utilizes a uniform packet format comprising a header, metadata, and $N = 16$ data payload flits. Each flit is 128 bits (16 bytes), enabling the transfer of $16 \times N = 256$ bytes of data per NVLink packet. The minimum data transmission size on NVLink-V1 is 32 bytes, corresponding to 2 data payload flits.

NVLink-V2 (GCP V100) and NVLink-V3 (GCP A100). However, the packet format for NVLink-V2 and V3 has not been documented. Therefore, we design experiments to reverse-engineer this format. Specifically, we initiate data transfers from GPU 1 to GPU 0 via NVLink using `cudaMemcpyPeer()`, varying the size of the data transferred. Concurrently, we profile the counter `nvlink_user_data_received` on GPU 0. This counter is chosen because it excludes headers and metadata, accurately monitoring the NVLink data transmission size.

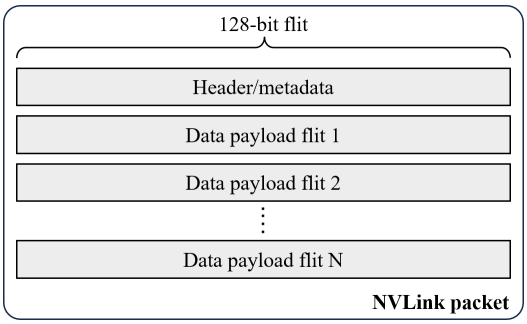


Figure 3: NVLink-V1 packet format.

Fig. 4 illustrates data flit usage with increasing transmissions on NVLink-V2 (GCP V100). We obtain three separate readings, as GPUs 0 and 1 each have three NVLink slots. As shown in Fig. 4(b), there are eight distinct steps within each 256-byte range. The three slots activate sequentially as transmission size grows: for sizes under 256 bytes, only slot 1 (blue) is active, while slots 2 and 3 remain idle; once the size exceeds 256 bytes, slot 2 (yellow) engages. This confirms that the maximum NVLink-V2 packet size, consistent with NVLink-V1, is 256 bytes, comprising 16 data payload flits.

Furthermore, Fig. 4(a) shows a zoomed-in view of the first NVLink packet. We observe eight steps over the 256-byte range, with each step corresponding to a 32-byte increase, or two data flits, indicating that the packet payload grows in two-flit increments. For instance, slot 1 (blue) uses two data flits (32 bytes) even when the transfer size is less than 32 bytes, suggesting this is the minimum transaction size. A similar pattern appears in NVLink-V3, as shown in Fig. 5, where we also observe 32-byte granularity.

These results reveal two behaviors. First, NVLink packetization is stable across generations: NVLink-V1/V2/V3 use a minimum transfer granularity of 32 bytes, increase in 32-byte increments, and cap each packet at 256 bytes. Second, for transfers larger than one packet, we observe behavior consistent with a dynamic multi-packet scheduler. Specifically, once the transfer size exceeds 256 bytes, multiple NVLink slots may become active, but the specific slot used for each subsequent packet can vary across runs. This suggests that NVIDIA may use an undocumented dynamic scheduling or traffic-scattering policy to distribute multi-packet traffic across available slots to exploit the available bandwidth.

> ***Observation 1****: The packetization granularity of NVLink-V1/V2/V3 is stable: the minimum data transmission size is 32 bytes, corresponding to two data payload flits, and the payload size increases in 32-byte increments. Each packet can carry up to 16 data payload flits, totaling 256 bytes. For multi-packet transfers, multiple slots are used concurrently, but the packet-to-slot assignment can vary across runs.*

Table 1: Specification of five target platforms.

	DGX-1 system	Server 1 (V100)	Server 2 (A100)	Server 3 (P100)	Server 4 (AMD)
CPU	Intel Xeon E5-2698v4	GCP N1-standard	GCP A2-highgpu	GCP N1-standard	AMD EPYC 7302
GPU	8 Tesla P100s	8 Tesla V100s	2 A100s (40 GB)	4 Tesla P100s (PCIe)	8 Radeon Instinct MI50
OS	Ubuntu 22.04	Ubuntu 20.04	Ubuntu 25.04	Ubuntu 20.04	Ubuntu 20.04
CUDA/ROCm	CUDA 12.2	CUDA 10.1.243	CUDA 12.8	CUDA 12.2	ROCm 6.1.1
GPU driver	535.129.03	525.125.06	570.172.08	535.261.03	6.2.4
NVLink/PCIe version	NVLink-V1	NVLink-V2	NVLink-V3	PCIe Gen3	PCIe Gen4
NVLink/PCIe slots	1 for peer GPUs	3 for peer GPUs	12 for peer GPUs	x16 per GPU	x16 per GPU

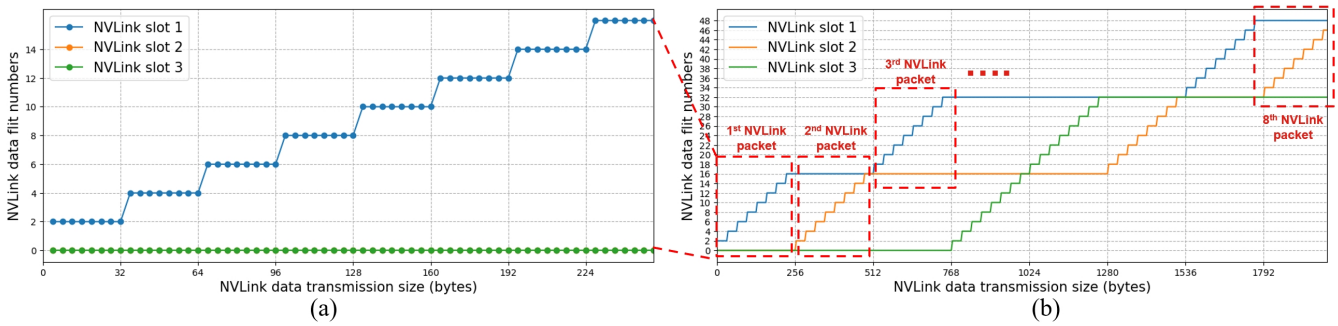


Figure 4: Reverse engineering NVLink transmission (NVLink-V2): (a) First packet, sizes 0–256 bytes, (b) Multiple packets.

Table 2: NVLink-related performance counters, available from NVIDIA CUPTI [50].

Category	Counter Name
Throughput	nvlink_receive/transmit_throughput
User	nvlink_user_data_received/transmitted, nvlink_user_write_data_transmitted, nvlink_user_response_data_received
Total	nvlink_total_data_received/transmitted, nvlink_total_response_data_received, nvlink_total_write_data_transmitted
Atomic operation	nvlink_total/user_nratom_data_transmitted, nvlink_total/user_ratom_data_transmitted

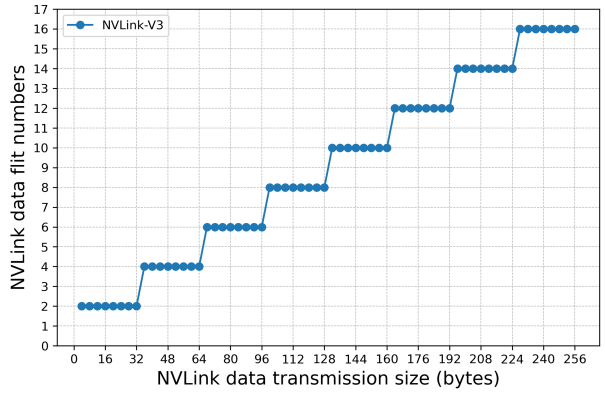


Figure 5: Reverse engineering NVLink-V3 (First packet).

3.2 NVLink Performance Counters

We collect all NVLink-related counters from CUPTI and evaluate them for potential leakage: Table 2 summarizes the 14 NVLink-related counters.

Experiment 1: Receive vs transmit. All NVLink counters feature both `receive` and `transmit` attributes. According to NVIDIA's white paper [22], an NVLink transaction begins with a request from the requester GPU, followed by the target GPU transmitting the data payloads back to the requester. The request consists only of metadata describing the requested data, making it significantly smaller than the payloads transmitted from the target GPU. We profile all NVLink counters by transmitting varying-sized packets using `cudaMemcpyPeer()`. We observe that on GPU 0 (receiver GPU), the `receive` counters consistently show significantly higher values than the `transmit` counters. Therefore, by examining NVLink receive/transmit values, the attacker can determine the victim's data transfer direction.

> ***Observation 2**: The NVLink **receive/transmit** attributes reveal NVLink data transaction direction.*

Experiment 2: User vs total. We observed that, aside from throughput, all categories feature both `user` and `total` attributes; these attributes are undocumented, so we explore them using the following experiment.

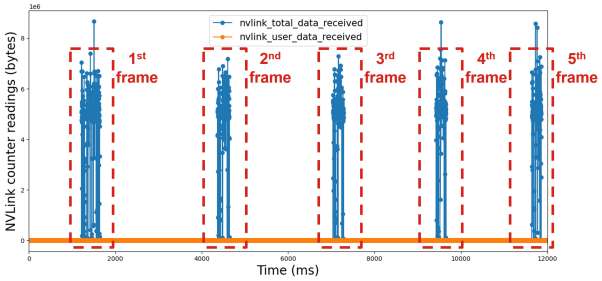


Figure 6: Counter traces for five consecutive frames.

In this experiment, a spy program continuously transfers data from GPU 0 to GPU 1 while profiling NVLink's `user` and `total` counters. Concurrently, we execute a program with a known NVLink communication pattern and observe the performance counters; in this experiment, we use a 3D rendering tool, Blender, to render 5 frames. Blender transfers 3D object data in each frame using the shared NVLink. As Fig. 6 shows, `user` counters remain unaffected. However, the `total` counters aggregate the transaction data from all programs using a shared NVLink. The five NVLink transmissions from Blender are clearly visible to the spy program. Therefore, by measuring the total NVLink values, the attacker can estimate the victim's data transmission size.

> ***Observation 3**: When NVLink is shared, the NVLink **total** counters reveal all data transaction patterns, including those from other users.*

Experiment 3: Aggregation mode. As illustrated in Fig. 1, P100 GPU (NVLink-V1) and V100 GPU (NVLink-V2) are equipped with 4 and 6 NVLink slots, respectively. NVLink offers an aggregate mode for performance counters that may be disabled [52]. We discovered that when aggregation mode is turned off, the counters detail the data transaction sizes for each individual NVLink slot. In the DGX-1 system, we obtain 4 values per GPU, each corresponding to one link. In contrast, for the GCP setup, we receive 6 values per GPU.

> ***Observation 4**: When aggregation mode is off, counter values are available for individual NVLink slots, providing finer-grained leakage.*

NVLink counter selection. As a result of these experiments, we identify two counters, `nvlink_total_data_received` and `nvlink_total_data_transmitted`, to use in our attacks. We rule out user counters since they do not leak information, while total counters leak transmitted data information regarding other applications. In addition, we observed that `throughput` counters incur higher overhead, limiting the rate of obtaining information. Finally, the counters tracking atomic operations remain zero for applications that do not use remote atomic operations, limiting their utility.

Potential generalization to other multi-GPU interconnects. We also observe that many performance counters available on AMD's Infinity Fabric [62] are similar to those on NVIDIA's NVLink. For example, the `L2-Fabric Read Bandwidth` counter [5] records the total number of bytes read by the L2 cache from the Infinity Fabric, which is analogous to `nvlink_receive_throughput` on NVIDIA GPUs. Similarly, the `Read/Write Latency` counter provides timing information on the number of cycles read/write requests spend in the Infinity Fabric before data is returned to the L2 cache. Recent research [25] has demonstrated that AMD's SEV does not disable performance counters when running an SEV-enabled VM, allowing CPU performance counters to be exploited for side-channel attacks by co-located processes. We believe that performance counters related to AMD's Infinity Fabric could similarly be leveraged to leak sensitive side-channel information.

Measurement overhead. We use the CUPTI API to collect performance counters on NVIDIA GPUs [46], which introduces runtime overheads that can be categorized into execution overhead and memory overhead. To minimize the impact on kernel concurrency, we set the data collection mode to continuous using `cuptiSetEventCollectionMode`. In this mode, CUPTI generates an instrumentation block when the CUDA module is loaded and applies it to the spy kernel at run-time. This instrumentation can add extra runtime overhead, which limits the sampling rate. For memory overhead, CUPTI uses static allocation by default, reserving three 3 MB pinned buffers per CUDA context at creation, which are freed when the context is destroyed. Additionally, the overhead of CUPTI depends on the number and type of profiling events and metrics collected. Profiling more counters increases the time required for data collection. We observe that NVLink-related counters incur overhead similar to other hardware components, such as SM-related or memory-related counters.

3.3 Timing-Based Leakage Due to Contention

The final set of reverse engineering experiments targets timing properties under contention. Contention occurs when multiple components simultaneously access shared resources, such as NVLink or PCIe, which are constrained by bandwidth or capacity limits.

3.3.1 Contention-Based Leakage on NVLink. In our threat model (Fig. 2), the victim program is on GPU A, and the spy program is on GPU B. The victim accesses data from GPU B via NVLink while the spy fetches data from GPU A, leading to contention on the Crossbar (XBAR), High-speed Hub (HSHUB), and NVLink I/O ports. The XBAR enables data exchange between GPU SM cores, L2 cache, and high bandwidth memory (HBM) [49]. The HSHUB connects the XBAR to the GPU's I/O ports (PCIe or NVLink) [47]. As they contend for these resources, this concurrent use of the NVLink from both programs results in observable delays compared to when the link is idle.

We use two programs in our experiments: (1) Program **A**, which continuously measures execution time in clock cycles using the `RDTSCP` instruction [32]. It does so for each `cudaMemcpyPeer()` execution (initiated by the CPU) that transfers 256 bytes (a single packet) via NVLink. (2) Program **B**, which transfers varying sizes of data over the shared NVLink, either explicitly via `cudaMemcpyPeer()` or implicitly through Unified Virtual Memory. We conduct experiments on both DGX-1 (NVLink-V1) and GCP machines (NVLink-V2 and V3).

Contention direction influence. Given that NVLink is bidirectional, we observe little to no delay when two programs transfer

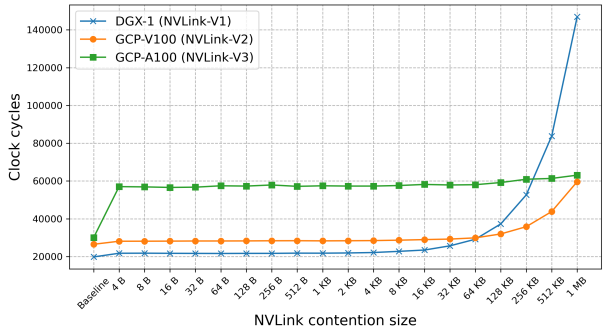


Figure 7: NVLink contention measurements.

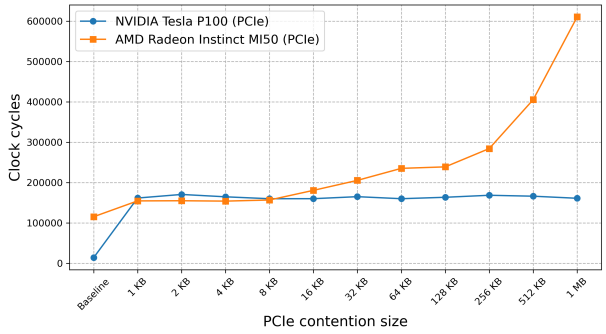


Figure 8: PCIe contention measurements.

data in opposite directions. In contrast, when both transfer in the same direction, contention occurs, leading to increased delay.

Contention size effect. Fig. 7 illustrates the impact of contention size on NVLink-V1/V2/V3 when program **B** transfers data through NVLink explicitly using `cudaMemcpyPeer()`. When two programs share NVLink, even transfers as small as 4 bytes cause measurable delays exceeding 10%. As contention size increases, timing delays rise accordingly. Beyond 256 bytes, NVLink-V1, V2, and V3 all exhibit consistent and significant contention. NVLink-V3 shows more stable delays because its 12 slots help relieve contention pressure. In contrast, NVLink-V2 (3 slots) and NVLink-V1 (1 slot) are more sensitive to contention.

3.3.2 Contention-Based Leakage on PCIe. Prior works [60, 67] demonstrate PCIe congestion-based leakages in CPU–GPU, CPU–SSD, and CPU–RDMA NIC settings. In this study, we show that such leakages also occur between GPUs. We conduct contention experiments on PCIe Gen3 (NVIDIA Tesla P100) and Gen4 (AMD Radeon Instinct MI50). As shown in Fig. 8, similar to our NVLink experiments, we observe little to no delay when two programs transfer data in opposite directions, due to PCIe's bidirectional design. However, delays increase proportionally as the contention size grows.

> ***Observation 5****: NVLink/PCIe contention arises when two transfers proceed in the same direction, leading to observable increases in data transfer time.*

Algorithm 1: Receiver for Covert Channel

```
// D_receiver[N] is an array of N bits to receive a message;
// T is the threshold to distinguish between bits '1' and '0';
Allocate an array R_0 in local GPU 0's memory;
Allocate an array R_1 in remote GPU 1's memory;
Synchronization();
for i ← 0 to N − 1 do
    Record time start via RDTSCP;
    Execute cudaMemcpyPeer() to copy R_1 to R_0;
    Record time end via RDTSCP;
    if end − start < T then
        D_receiver[i] ← 0;
    else
        D_receiver[i] ← 1;
    end
end
```

4 Attacks in the GPU Box

In this section, we explore contention-based covert and side-channel attacks in the GPU box. The attacker leverages the ability to measure NVLink timing contention without requiring superuser privileges.

4.1 Covert-Channel Attacks via Contention

Synchronization. Synchronization is essential for improving the bandwidth and controlling the error rate on a covert channel. We use a three-phase synchronization protocol for the covert channel. The sender first transmits a pre-agreed data pattern consisting of a long sequence of continuous high values followed by a short sequence of continuous low values to signal readiness. The receiver then replies with a corresponding pattern to acknowledge readiness. Once the handshake completes, the sender transmits the message bits. To reduce errors, the sender prepends the same pattern to the message, allowing the receiver to detect the start of transmission.

Covert-channel design: ContenLink. As we observed in Section 3, contention on a shared NVLink leads to an increase in data transfer time. We build a covert channel that exploits this timing difference to transmit data covertly. Algorithm 1 outlines the design of the *ContenLink* receiver. Initially, the receiver allocates two 256-byte arrays, R_0 in the local GPU 0 and R_1 in the remote GPU 1, respectively. We set the array size to 256 bytes because, based on our first observation, this is the maximum size of a single NVLink packet. The receiver measures the execution time for invoking the `cudaMemcpyPeer()` API, which facilitates the copying of arrays R_1 to R_0 via NVLink, using the hardware timer accessible through the `RDTSCP` instruction. If the measured time falls below a threshold T, the receiver interprets this as free of contention on the NVLink, indicating a received bit of '0'. Conversely, a measurement exceeding the threshold signifies contention, interpreted as a bit of '1'.

Algorithm 2 presents the sender's design for the covert channel. The initial step involves allocating two arrays, S_0 and S_1, on two GPUs. The differing sizes of these arrays are intended to exert varying levels of contention pressure on the NVLink. We evaluate the impact of varying sender sizes on the covert channel's bandwidth and error rate. After synchronizing with the receiver, the sender starts the transmission of covert messages. To transfer a bit '1', it uses `cudaMemcpyPeer()` API to copy S_1 to S_0 from remote GPU to local GPU, creating contention on the shared NVLink with

Algorithm 2: Sender for Covert Channel

```
// D_sender[N] is an array of N bits used to send a message;
// K is the number of executions of NOPs;
Allocate an array S_0 in local GPU 0's memory;
Allocate an array S_1 in remote GPU 1's memory;
Synchronization();
for i ← 0 to N − 1 do
    if D_sender[i] == 1 then
        Execute cudaMemcpyPeer() to copy S_1 to S_0;
    else
        for j ← 0 to K − 1 do
            Execute a NOP;
        end
    end
end
```

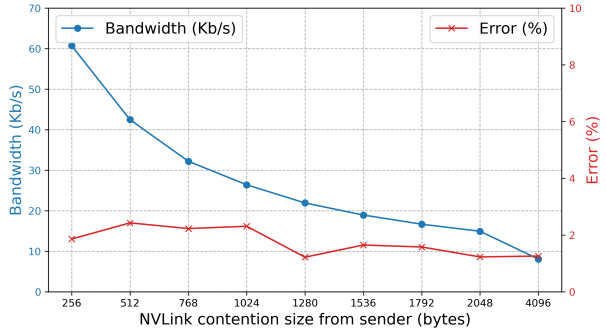


Figure 9: DGX-1 system (NVLink-V1) contention covert channel results.

the receiver. For transmitting a bit '0', it performs K loops of NOP instructions, which do not cause contention on the NVLink.

Evaluation of *ContenLink*. We transmitted a 10,000-bit message five times and calculated the average bandwidth and error rate. The error rate was determined using the Levenshtein edit distance [40]. Each message, composed of an equal number of '0' and '1' bits, was randomly generated. Fig. 9 and Fig. 10 illustrate the bandwidth and error rate on two platforms. Based on our contention measurements detailed in Section 3.3, the varying data pressures exerted by the sender distinctly affect the covert channel's performance. We tested sender data sizes ranging from 256 bytes to 4 KB. We ignore sizes smaller than 256 bytes because we find that when the contention size is small, it becomes challenging for the receiver to distinguish between bit '1' and '0', resulting in a high error rate. On the GCP-V100 machine, we achieved the highest bandwidth of 70.59 kbps with a sender data size of 256 bytes, accompanied by an error rate of 4.78%. Similarly, on the DGX-1 machine, the highest bandwidth reached was 60.71 kbps with the same sender data size, resulting in a lower error rate of 1.86%. As the data pressure from the sender increases, we observe a corresponding decrease in the bandwidth of covert channels on two platforms, although the error rate remains stable.

4.2 Side-Channel Attacks via Contention

In this section, we demonstrate two end-to-end side-channel attacks: application fingerprinting and 3D object fingerprinting attacks.

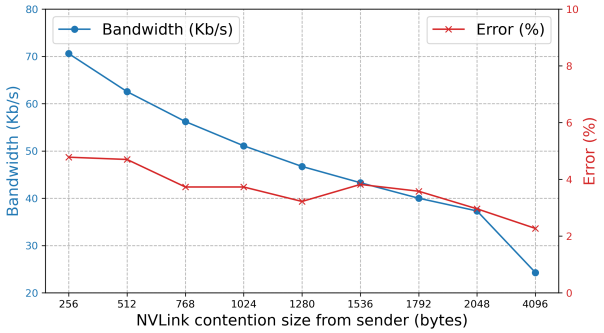


Figure 10: GCP V100 (NVLink-V2) contention covert channel results.

4.2.1 Attack 1: Application Fingerprinting. In this attack, we reveal that an adversary can infer the specific HPC applications or deep learning models by exploiting NVLink side-channel leakages. We test application fingerprinting attacks on 8 HPC applications from the OpenMM [19] benchmarks and 10 deep-learning models as the victim applications.

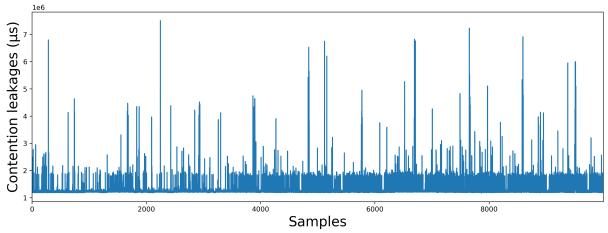
OpenMM benchmark. OpenMM is a high-performance toolkit tailored for molecular dynamics simulations, supporting multi-GPU systems. In this study, we center our attention on 8 benchmark applications [69]: rf, pme, apoa1-rf, apoa1-pme, apoa1-ljpme, amoeba-pme, amber20-dhfr, and amber20-cellulose.

Deep learning models. In this work, we consider the attacker's attempts to extract the model structure like in previous work [16, 42, 67, 75]. Specifically, we select 10 popular deep learning models (5 simple and 5 complex). The simple models include a Multi-layer Perceptron (MLP), two basic Convolutional Neural Networks (CNNs), a regression model, and a Long Short-Term Memory (LSTM) network [29]. For the complex models, we chose five famous models: AlexNet [36], VGG16 [61], GoogLeNet [66], and two variants of ResNet (ResNet-18 and ResNet-50) [28] used in Computer Vision (CV). Each model was trained on the MNIST dataset [14], utilizing PyTorch-supported data parallelism for distributed training across 100 iterations [54]. The batch size was set at 64. For complex models, the image size was resized to 224 by 224. We leave the layer hyper-parameter extraction as future work.

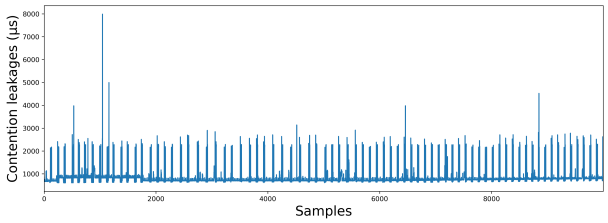
Experimental setup and data collection. We evaluate the attack on five testbeds covering NVLink V1, V2, V3, and PCIe. We assume the victim executes their application on multiple GPUs, while the spy executes on another GPU that shares the same NVLink or PCIe interconnect. The spy runs in the background and collects timing delays reflecting the presence or absence of contention from the victim's traffic. During the data collection phase, we collected 50 traces of side-channel leakages for each application. The dataset was divided into training and testing sets using an 80/20 split ratio.

Observing benchmark distinguishability. Fig. 11 shows traces for three applications, amber20_cellulose, AlexNet and ResNet-50, highlighting their distinguishable characteristics. The X-axis represents profiling samples, while the Y-axis shows contention timing leakages.

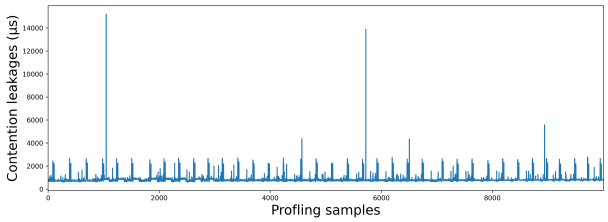
Feature engineering and classification. We extract features from the collected time-series data. We use a sliding window approach



(a) The NVLink leakage traces of amber20_cellulose.



(b) The NVLink leakage traces of AlexNet.



(c) The NVLink leakage traces of ResNet-50.

Figure 11: Comparison of NVLink leakage traces.

with a window size of 100 to compute 12 statistical features, listed in Table 3. We choose a window size of 100 samples to balance temporal locality and statistical stability: smaller windows preserve finer timing variations but produce noisier feature estimates, while larger windows filter out short leakage patterns, reducing accuracy. The selected features capture complementary statistical properties of each window, including central tendency (*mean*, *median*), extrema (*max*, *min*), dispersion (*std*, *var*, *range*, *iqr_val*), aggregate magnitude (*sum*), tail/distribution information (*percent_25*, *percent_75*), and burst intensity (*count_am*). For *count_am*, the threshold is computed as the mean value within the same window, avoiding the use of information from other traces. We keep the sampling rate fixed within each dataset before feature extraction so that windows correspond to comparable temporal spans across traces.

We then use the extracted features to build classification models, employing three standard machine learning algorithms: K Nearest Neighbors (KNN) [35] with parameter settings: $n\_neighbors = 5, leaf\_size = 30$, XGBoost [8] with parameter settings: $n\_estimators = 100, max\_depth = 6$, and Light Gradient Boosting Machine (LightGBM) [34] with parameter settings: $num\_leaves = 31, max\_depth = -1, n\_estimators = 100$. To assess classifier performance, we report macro-averaged precision, recall, and F1 score [30]. For each class c, we compute $P_c = TP_c/(TP_c + FP_c)$, $R_c = TP_c/(TP_c + FN_c)$, and $\mathrm{F1}_c = 2P_cR_c/(P_c + R_c)$. We then average each metric equally across the C classes; for example, $\mathrm{F1}_{\mathrm{macro}} = \frac{1}{C}\sum_{c=1}^{C}\mathrm{F1}_c$.

Table 3: Definitions of statistical features used in *NVBleed*.

Features	Description
mean	Mean of all values.
max	Maximum of all values.
min	Minimum of all values.
median	Median of all values.
std	Standard deviation.
var	Variance.
range	Difference of maximum and minimum.
sum	Sum of all values.
count_am	Count of observations that exceed the mean.
percent_25	25th percentile value.
percent_75	75th percentile value.
iqr_val	Interquartile range.

Table 4: Application fingerprint performance: F1 (%), Precision (%), and Recall (%) on DGX-1 and GCP V100.

	DGX-1 (NVLink-V1)			GCP V100 (NVLink-V2)		
	F1	Prec	Rec	F1	Prec	Rec
KNN	36.22	37.26	39.44	55.77	55.97	58.89
XGBoost	**82.69**	82.95	82.78	96.65	96.90	96.67
LightGBM	81.21	81.25	81.67	**96.77**	97.23	96.67

Table 5: Application fingerprint performance: F1 (%), Precision (%), and Recall (%) on GCP A100 (NVLink-V3) and GCP P100 (PCIe Gen3).

	GCP A100 (NVLink-V3)			GCP P100 (PCIe Gen3)		
	F1	Prec	Rec	F1	Prec	Rec
KNN	73.56	75.33	74.17	64.91	66.89	65.71
XGBoost	89.02	89.62	89.17	**75.67**	76.19	75.71
LightGBM	**89.86**	90.46	90.00	74.19	74.47	74.29

Results. Table 4 presents the classification performance results. On the DGX testbed and GCP V100 machine, XGBoost and LightGBM achieve the highest accuracy, with F1 scores exceeding 82% and 96%, respectively, using timing leakage. Table 5 reports the results for the GCP A100 and GCP P100 platforms, where LightGBM and XGBoost achieve the best performance, with F1 scores of 89.86% and 75.67%, respectively. On the AMD PCIe platform, XGBoost and LightGBM also perform well, reaching F1 scores of 80.80% and 77.94%, respectively, while KNN lags behind at 48.72%.

Sensitivity to input sizes. We explore whether application fingerprinting can be successful when the input sizes change in the context of Deep Learning models. Although the side-channel signal values change with input size, we find that the leakage patterns remain identifiable with different input image sizes (for an image classifier). To analyze this effect, we collected additional side-channel data that includes variable input sizes of 32, 64, and 128 for model training. We applied the same feature engineering and classification setup. On the GCP V100 with 2 GPUs, the best F1 score decreases slightly from 96.77% to 96.48% with XGBoost. With 4 GPUs, it drops further to 83.05% (XGBoost), and with 8 GPUs, it declines to 81.88% (LightGBM).

Sensitivity to GPU topology. Victims may use multi-GPU topologies beyond just two GPUs, such as 4-GPU or 8-GPU configurations. Our evaluation shows that in 4-GPU setups, under timing leakage, LightGBM delivers the best performance on DGX-1 (F1 = 88.04%), while XGBoost slightly outperforms others on GCP V100 (F1 = 89.17%). In 8-GPU configurations, LightGBM achieves the highest F1 scores on both DGX (82.27%) and GCP V100 (73.41%). Overall, we observe that as GPU topologies grow more complex, attack accuracy decreases but remains effective.

4.2.2 Attack 2: Identifying Rendered Characters. We present a proof-of-concept attack on the Blender rendering tool, showing that workload-dependent NVLink traffic patterns can reveal rendering properties. Specifically, we evaluate whether an attacker can identify which 3D character is being rendered among a fixed set of candidate characters. We first provide background on how Blender renders in a multi-GPU setup, and then describe the attack methodology and evaluation results.

Attack rationale. The goal of this experiment is to evaluate whether application-level workload differences can induce distinguishable inter-GPU communication patterns. This attack is more specific than application fingerprinting: instead of identifying which application is running, the attacker infers coarse-grained rendering properties within a known Blender workload. The attack can only recover the communication behavior, and therefore can only gain visibility into information that affects the communication behavior. With respect to Blender, under controlled settings, different rendered characters can produce distinguishable NVLink traces. Such leakage may pose security or privacy risks in contexts where rendered graphical features are sensitive, such as avatars that indicate a user's presence in a virtual space or representations that resemble personal traits such as facial features, body shape, or movement patterns [39, 70].

3D character rendering on multi-GPU systems. 3D scene rendering is an intensive process utilized to generate images and movies from scenes modeled in specialized software environments, such as Blender [11], Unity [27], Unreal Engine [24], and others. Utilizing GPUs to accelerate 3D rendering has become a standard practice within these toolkits [33]. The initial step in rendering 3D scenes with GPU assistance involves transferring scene data from CPU to GPU memory. After the computation is completed, the rendered data is transferred back to the CPU to finish rendering. In this study, we choose Blender as the target 3D rendering toolkit for two main reasons: first, its widespread use and open-source nature; second, its support for accelerated rendering on multiple GPUs through NVLink. By activating the "Distribute Memory Across Devices" option in Blender, the 3D scene data is not duplicated across each GPU. Instead, this data is first transferred to a single GPU. Subsequently, all NVLink-connected peer GPUs can share the scene data efficiently. This approach significantly enhances the speed of rendering complex and massive scenes.

Experimental setup and data collection. Similar to the application fingerprinting attack, the spy executes on a GPU that shares the NVLink interconnect with the victim's multi-GPU application. The background spy program profiles timing delays resulting from the victim application's contention on NVLink. Our evaluation of the attack was conducted on a GCP 2-V100 instance. We discovered that Blender does not support configurations with 4-V100 or 8-V100 on GCP, as each pair of GPUs requires an NVLink connection. However, as depicted in Fig. 1b, the GCP GPU machine's ring topology does not provide the required NVLink connectivity for 4- or 8-GPU Blender configurations. For victim 3D characters, we select 50 fully rigged characters from the Blender Studio open movies [12]. Each character is rendered within the same background scene with the same camera angle and resolution. We split the dataset into training and testing using an 80/20 ratio.

Table 6: 3D graphics character fingerprint performance: F1 (%), Precision (%), and Recall (%) on GCP V100 (NVLink-V2).

	GCP V100 (NVLink-V2)		
	F1	**Prec**	**Rec**
KNN	61.37	63.21	64.50
XGBoost	85.30	87.70	85.50
LightGBM	**86.86**	88.99	87.00

Feature engineering and classification. Similarly, we exploit a sliding window approach with a window size of 200 to extract the same 12 time-series features as our application fingerprinting attack. We then use KNN, XGBoost, and LightGBM models to infer which 3D character the victim renders. As shown in Table 6, LightGBM obtains the highest F1 score (86.86%), with XGBoost following closely at 85.30%.

Limitations of the side-channel attacks. Our end-to-end leakage demonstration is limited to Blender rendering workloads under controlled experimental settings. Our results show that application behaviors that change inter-GPU communication patterns can induce distinguishable NVLink timing traces; only these behaviors are vulnerable to this attack. In the Blender case study, these traces are sufficient to infer coarse-grained rendering properties. Extending this analysis to other multi-GPU applications and richer victim workloads is an important direction for future work.

5 Attacks on the Cloud across VM

During reverse engineering, we made a surprising observation: NVLink counters leak information across GPUs that are not directly communicating, as long as a connecting (but unused) link exists. This potentially enables GPUs belonging to different VM instances to observe leakage across instances. In this section, we first demonstrate and characterize this leakage between two VM instances on GCP. We then implement the 3D rendering fingerprinting side-channel attack across the two VM instances using this leakage.

Prerequisite for this attack. The prerequisite for this attack is co-location, meaning the attacker must reside on the same physical host as the target victim. Previous research has extensively demonstrated co-location attacks in cloud systems [20, 56, 82, 83]. Specifically, Zhao et al. [82, 83] presented lightweight and highly successful co-location attacks on Google Cloud Run. Based on these findings, we assume that the attacker has successfully co-located on the same physical GPU host/node as the victim. The attacker then exploits NVLink leakage to infer victim workload information.

Experiment 4: Cross-VM leakage on GCP. We consider a co-located cross-VM setting where the attacker and victim are on the

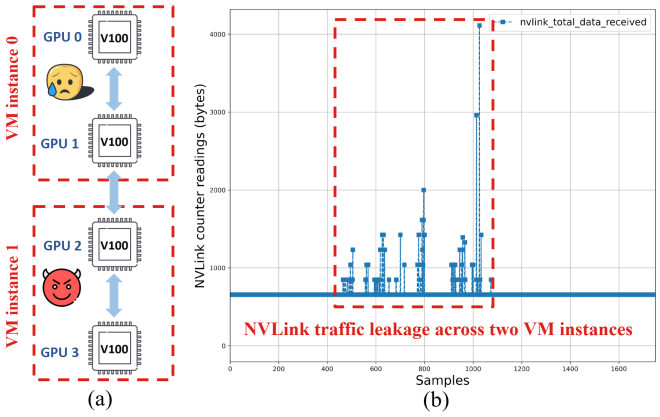


Figure 12: Cross-VM Leakage: (a) Instance topology: Victim resides in VM Instance 0, while the spy operates in Instance 1; (b) NVLink leakage collected on GPU 2 by the spy VM across the two VM instances.

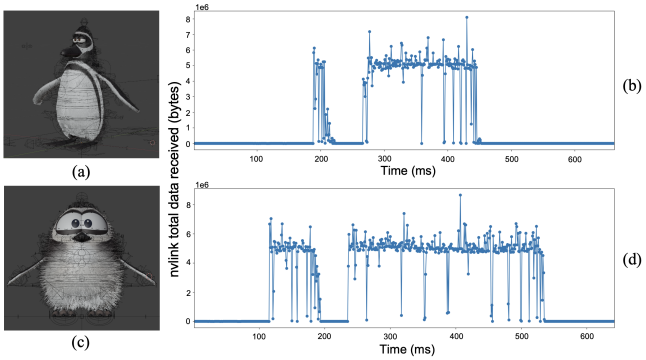


Figure 13: The NVLink leakage traces for two 3D characters from Blender Studio: (a) Character 1: Pinguino [65], (b) the NVLink leakages for Pinguino, (c) Character 2: Oti [64], and (d) the NVLink leakages for Oti.

same physical host. We obtain a 4-V100 GPU instance on GCP and create two KVM VMs on an Ubuntu 20.04 host, assigning two GPUs to each VM. This setup reflects a practical case where VMs are logically isolated, but their GPUs remain connected through the shared NVLink fabric; the exposed NVLink links and interconnect-level counters are not fully isolated or virtualized. As shown in Fig. 12(a), the victim's VM instance (Instance 0) includes GPUs 0 and 1, while the spy's VM instance (Instance 1) contains GPUs 2 and 3. NVLink is used as the interconnect in this topology. All experiments are conducted on GCP Compute Engine in the us-west1-a region within a controlled environment that does not affect external public users.

In this experiment, the victim uses Blender to render 2D images from 3D characters. The 3D object data is transmitted between GPU 0 and GPU 1 via NVLink. Simultaneously, the spy in another VM instance monitors the NVLink counter (`nvlink_total_data_received`) on GPU 2. As shown in Table 1, each V100 GPU is equipped with six NVLink slots, three of which are connected to peer GPUs. This allows GPU 2 in the spy's VM instance to monitor NVLink traffic patterns on the link between GPU 1 and GPU 2.

We observe that when two VM instances are connected, the spy can track NVLink traffic patterns from another VM instance by observing performance counters. Figure 12(b) depicts the NVLink traffic traces collected across the two VM instances. The signal measured on GPU 2 does not reflect the full traffic between GPU 1 and GPU 0, but it is correlated with that traffic and provides consistent patterns for different Blender characters. The spikes in the middle of the trace correspond to the victim's rendering execution in VM Instance 0.

> ***Observation 6**: When two VM instances are connected via NVLink, NVLink counters leak traffic patterns across VM instances even though this traffic does not occur on the physical link connecting the instances.*

Attack 3: Identifying rendered 3D characters across VMs. In this attack, we investigate whether the attacker can identify the 3D character the victim is rendering in a cross-VM scenario. Similar to our second attack, the background spy program monitors specific performance counters, including `nvlink_total_data_received`. The victim renders 50 fully rigged 3D characters from the Blender Studio open movies. Each character is rendered in the same background scene, using the same camera angle and resolution. The dataset is split into training and testing sets using an 80/20 ratio.
Observing 3D character distinguishability. Fig. 6 depicts the NVLink counter readings for five consecutive frames. These readings clearly show the start and end times of rendering for each frame. When zoomed in, as illustrated in Fig. 13, we observe that the renderings of two 3D characters (Character 1: Pinguino [65] and Character 2: Oti [64]) exhibit distinct NVLink transfer patterns.
Feature engineering and classification. Similarly, we use a sliding window approach with varying window sizes ranging from 100 to 1000 to extract the same 12 time-series features as in our application fingerprinting attack. We then apply KNN, XGBoost, and LightGBM models (using the same hyperparameter settings as in Attack 2) to infer which 3D character the victim is rendering. We observe that with a window size of 1000, LightGBM achieves the best performance, with an F1 score of 88.57%, a precision of 91.16%, and a recall of 88.50%.

6 Potential Mitigations

We discuss four potential classes of mitigations: (1) managing access to leaky counters, (2) restricting access to high-resolution clock timing instructions, (3) detecting abnormal NVLink monitoring and/or contention, and (4) injecting noise into the NVLink performance counters. Counter scoping, sampling-rate limits, and noise injection primarily target counter-based leakage. In contrast, restricting high-resolution timers, detecting abnormal contention, and enforcing scheduling or resource isolation target timing-based contention leakage by limiting measurement or enforcing isolation.
Managing access to counters. A natural mitigation is to scope NVLink performance counters to the requesting GPU, process, or VM. Ideally, a process or VM should only observe counter values

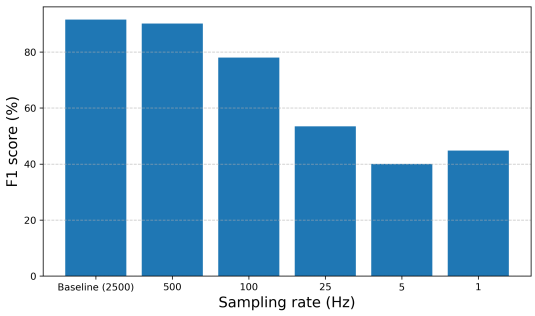


Figure 14: F1 score under reduced sampling rate.

generated by its own execution context, and counter state should be cleared, virtualized, or filtered across context switches and VM boundaries. This would directly reduce the cross-GPU and cross-VM counter-based leakage observed in our attacks. We believe such isolation may be feasible in principle, but its exact implementation requires vendor support because the counter collection and exposure paths are not publicly documented.

Counter scoping only addresses the counter-based leakage but does not eliminate the timing-based contention channel, where an attacker may still infer victim activity by measuring delays caused by shared NVLink or interconnect resources. Defending against this channel requires separate mechanisms, such as interconnect scheduling isolation, traffic shaping, or stronger resource partitioning, which may reduce utilization or introduce performance overhead.

Reducing the sampling rate of performance counters may diminish the effectiveness of the spy application [42, 79]. However, this approach could negatively impact legitimate users, reducing the visibility into program behavior. Fig. 14 presents the impact of reducing the sampling rate on 3D character classification. Although the sampling rate is reduced to 1 Hz, the attacker can still gain information about the 3D character with an F1 score exceeding 40%, significantly outperforming a random guess (2%). This suggests that sampling-rate reduction alone may not be sufficient to close the channel.

Restricting access to high-resolution clock instructions. Our attacks utilize the high-resolution clock instruction, *RDTSCP*, to measure the victim's contention on a shared NVLink. Completely blocking access to such instructions significantly interferes with the attack; however, legitimate users may also require such instructions to evaluate the performance of their programs. On the other hand, even if specific timing instructions are blocked, attackers can still utilize alternative low-requirement clock instructions (*e.g.*, Timer interrupts [57]) or create a synthetic timer [16, 58]. Therefore, timer restriction can raise the attack cost, but it is unlikely to eliminate the leakage by itself, especially when counter-based signals or alternative timing sources remain available.

Detecting abnormal NVLink monitoring and/or contention. *NVBleed* operates by creating contention on shared NVLink interconnects and continuously probing the NVLink. A potential defense involves monitoring for such suspicious NVLink querying behaviors. GPUGuard [76] proposes a contention detection and dynamic partitioning framework. However, their method only protects single-GPU applications and is incompatible with multi-GPU systems, and current NVLink hardware does not support dynamic partitioning. More generally, contention-based defenses must distinguish malicious probing from legitimate high-throughput multi-GPU communication, which may be challenging for performance-sensitive workloads.

Adding noise to obscure performance counters. Differential Privacy (DP) has received considerable attention as a method for mitigating side-channel leakages to obscure sensitive user data attributes [7, 43]. In our work, we quantitatively obscure sensitive NVLink performance counters by adding noise to maximize privacy while minimizing the impact on the usability of the original counter readings. Noise-based defenses introduce an accuracy–utility trade-off: larger noise provides stronger privacy and reduces attack accuracy, but also reduces the fidelity of counter readings for legitimate profiling. Moreover, if noise is added independently to each sample, repeated sampling may allow attackers to partially average out the noise. Thus, noise-based defenses should ideally be combined with a sampling budget or rate limit. Specifically, we utilize the Bounded Laplace Mechanism [17], which addresses issues associated with unbounded noise that can produce semantically invalid results (*e.g.*, negative NVLink performance counter values). The bounded mechanism adjusts the noise distribution using the privacy parameters and a fixed value, then samples until the output falls within specified bounds. In our setup, we define the meaningful range for the NVLink counter to be between 0 and 1 MB to ensure the utility of the counter remains intact. The level of privacy provided by the defense depends on the appropriate selection of the differential privacy parameter ε, where a smaller ε adds more noise and yields a stronger privacy guarantee. We add DP noise to the raw NVLink performance counter readings collected during Attack 3, which aims to infer rendered 3D characters among 50 classes. We use the same feature engineering and classification settings as in the original attack evaluation. Figure 15 shows the classifier's performance under DP budgets (ε). As ε decreases, the added DP noise increases, progressively distorting the performance counter signals. This obfuscation reduces the classifier's ability to accurately distinguish traces, resulting in a steady decline in F1 score. At low ε values (e.g., 0.1 or 0.05), the fingerprinting success rate approaches that of random guessing.

Overall, these mitigations provide complementary protection. Counter scoping and virtualization are the cleanest defenses against cross-GPU and cross-VM counter leakage, but they require vendor-side support. Sampling-rate limits and DP-style noise can be implemented as lighter-weight defenses, but they trade off profiling utility and may not fully eliminate leakage. Most importantly, these counter-focused defenses do not remove timing-based contention leakage, which requires separate resource-isolation or scheduling-based mechanisms. A complete defense therefore likely requires combining counter isolation with restrictions on high-rate monitoring and stronger isolation of shared interconnect resources.

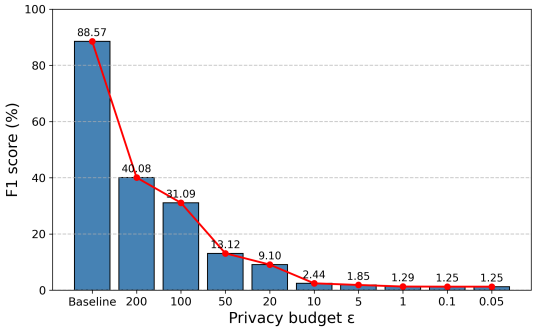


Figure 15: F1 score with differential privacy noise.

7 Related Work

GPUs are widely deployed in HPC and cloud infrastructures to accelerate diverse workloads, but prior work shows that they also introduce new security vulnerabilities.

Covert and side-channel attacks in GPUs. Covert and side-channel attacks have been extensively studied in discrete GPUs. Naghibijouybari et al. [41] introduced a contention-based covert channel across various GPU resources. Subsequent work conducted multiple side-channel attacks on both graphics and computational GPU workloads by monitoring GPU performance counters [42, 75]. Nazaraliyev et al. [45] showed covert and side-channel attacks on consumer GPUs' memory by exploiting the Rowhammer mitigation mechanism. Ahn et al. [1] proposed a timing covert channel for GPUs that exploits the shared, on-chip interconnect channels. Additionally, Nayak et al. [44] developed a covert channel leveraging the GPU's shared last-level translation lookaside buffer (TLB). Recent studies have also highlighted several covert and side-channel attacks on integrated GPUs. Dutta et al. [15] demonstrated covert and side-channel attacks between CPUs and GPUs through the shared Last Level Cache (LLC) and ring bus in integrated CPU-GPU systems. Yang et al. [77] present a side-channel attack on mobile GPUs to eavesdrop on user credentials via GPU performance counters. Almusaddar et al. [2] exploited the observable slowdown in shared buffers within the memory controller and constructed a covert channel in integrated CPU-GPU systems. Wang et al. [73] identified side-channel leakage during the graphical data compression process on integrated GPUs. Taneja et al. [68] demonstrated a web fingerprinting attack by probing power, temperature, and frequency on integrated GPUs. Another study by Ferguson et al. [21] introduced WebGPU-SPY, a GPU-based cache side-channel attack on integrated GPUs.

Interconnect contention-based attacks. Our attacks can be classified as interconnect contention-based attacks because they exploit contention on shared NVLink and PCIe communication paths. Prior work has shown that contention on shared interconnects can be used to construct covert and side channels. For example, Tan et al. [67] demonstrated PCIe congestion side channels across GPUs, Network Interface Cards (NICs), and Solid-State Drives (SSDs). Other attacks have exploited contention on shared hardware resources and interconnects, including the last-level cache [38], CPU ring bus [53], Host-GPU PCIe bus [60], CPU mesh interconnect [72], and file-system synchronization primitives [26]. NVBleed belongs to this class of side-channel attacks using the same insight that traffic from different sources on a shared communication fabric can induce measurable contention.

NVBleed differs from existing interconnect side-channel attacks in both target system and leakage semantics. Prior interconnect attacks primarily exploit on-chip processor interconnects or CPU-peripheral buses. In contrast, NVBleed targets multi-GPU interconnects, including NVLink and PCIe GPU-to-GPU paths. This setting introduces a different topology and isolation model: the victim and spy do not need to share the same GPU or processor. Instead, the spy can reside on a different GPU and, in some settings, even in a separate VM instance, while still observing leakage associated with the victim's inter-GPU communication. The attacks differ in terms of the target leakage, with prior interconnect attacks typically exploiting traffic related to cache accesses, memory transactions, or general I/O activity. NVBleed exposes a different leakage source: GPU-to-GPU communication behavior. Rather than inferring fine-grained memory access patterns, our attack reveals coarse-grained information about when victim workloads communicate across GPUs and how their inter-GPU traffic patterns change over time. This communication-level leakage is especially relevant to modern multi-GPU workloads, where inter-GPU transfers can reflect workload structure, synchronization behavior, and workload-dependent execution phases.

The closest prior work in multi-GPU systems is Dutta et al. [16], which uses prime-and-probe to create cache contention via pinned memory pages. NVBleed targets a different shared resource and leakage signal, thereby exposing a complementary leakage source. Dutta et al. infer memory access behavior through multi-GPU cache sharing effects, whereas NVBleed infers inter-GPU communication behavior through contention on the communication fabric itself. By demonstrating end-to-end side- and covert-channel attacks over NVLink and PCIe, including cross-GPU and cross-VM leakage, NVBleed extends interconnect side-channel research beyond single-chip fabrics, CPU-peripheral buses, and cache/memory-based multi-GPU attacks. To the best of our knowledge, this is the first work to exploit multi-GPU interconnect communication as a source of side- and covert-channel leakage.

Cross-VM attacks. Several prior studies have demonstrated cross-VM attacks on cloud systems. Ristenpart et al. [56] conducted the first study on co-residence detection in commercial cloud servers. Fang et al. [20] showed how attackers could manipulate cloud schedulers to achieve high co-location rates with target victims. Recently, Zhao et al. [82, 83] conducted co-location attacks in public cloud systems and successfully implemented an end-to-end LLC Prime+Probe side-channel attack on Google Cloud Run.

8 Concluding Remarks

We present covert and side-channel attacks on multi-GPU interconnects, including NVLink and PCIe. By reverse-engineering three NVLink generations, we identify timing- and counter-based leakage and demonstrate four attacks: one covert channel, two GPU-box side channels, and one cross-VM side channel. We also evaluate defenses, including differential privacy noise.

Acknowledgments

This work was supported in part by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Award 66150, "CENATE - Center for Advanced Technology Evaluation," and in part by the National Science Foundation (NSF) under Grants CNS-2448156, CCF-2212426, and CNS-2053383.

Ethical Considerations

All experiments in this work were conducted on controlled private testbeds within our laboratory or in isolated cloud sandboxes. At no point did our research involve unsuspecting human participants, nor did it interact with or impact other users or production systems. We responsibly disclosed our findings to NVIDIA and Google.

Open Science

We commit to full compliance with the ACM CCS Open Science Policy. All artifacts are available at https://anonymous.4open.science/r/nvbleed-410D/, including covert-channel code, side-channel traces, analysis notebooks, datasets, and trained models.

References

[1] Jaeguk Ahn, Jiho Kim, Hans Kasan, Leila Delshadtehrani, Wonjun Song, Ajay Joshi, and John Kim. 2021. Network-on-chip microarchitecture-based covert channel in gpus. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 565–577.

[2] Ghadeer Almusaddar and Hoda Naghibijouybari. 2023. Exploiting parallel memory write requests for covert channel attacks in integrated CPU-GPU systems. *arXiv preprint arXiv:2307.16123* (2023).

[3] Ghadeer Almusaddar, Yicheng Zhang, Saber Ganjisaffar, Barry Williams, Yu David Liu, Dmitry V. Ponomarev, and Nael B. Abu-Ghazaleh. 2025. ShadowScope: GPU Monitoring and Validation via Composable Side Channel Signals. *CoRR* abs/2509.00300 (2025). arXiv:2509.00300 doi:10.48550/ARXIV.2509.00300

[4] AMD. 2023. AMD uProf. https://www.amd.com/en/developer/uprof.html#documentation.

[5] AMD. 2025. ROCm Compute Profiler. https://rocm.docs.amd.com/projects/rocprofiler-compute/en/latest/index.html#.

[6] Ravi Budruk, Don Anderson, and Tom Shanley. 2004. *PCI express system architecture*. Addison-Wesley Professional.

[7] Joann Qiongna Chen, Tianhao Wang, Zhikun Zhang, Yang Zhang, Somesh Jha, and Zhou Li. 2023. Differentially private resource allocation. In *Proceedings of the 39th Annual Computer Security Applications Conference*. 772–786.

[8] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proc. ACM SIGKDD Conf. on Knwl. Discov. and Data Min. (KDD)*. 785–794.

[9] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 tensor core GPU: Performance and innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[10] Jack Choquette, Olivier Giroux, and Denis Foley. 2018. Volta: Performance and programmability. *IEEE Micro* 38, 2 (2018), 42–52.

[11] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

[12] Blender Online Community. 2023. *Blender Studio Character library*. https://studio.blender.org/characters/

[13] John Danskin and Denis Foley. 2016. Pascal GPU with NVLink. In *2016 IEEE Hot Chips 28 Symposium (HCS)*. IEEE Computer Society, 1–24.

[14] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[15] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2021. Leaky buddies: Cross-component covert channels on integrated CPU-GPU systems. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 972–984.

[16] 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*. 1–13.

[17] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[18] Doug Eadline. 2024. Everyone Except NVIDIA Forms Ultra Accelerator Link (UALink) Consortium. *HPCwire* (May 30 2024). Accessed: 2024-06-20. https://www.hpcwire.com/2024/05/30/everyone-except-nvidia-forms-ultra-accelerator-link-ualink-consortium/

[19] Peter Eastman, Jason Swails, John D Chodera, Robert T McGibbon, Yutong Zhao, Kyle A Beauchamp, Lee-Ping Wang, Andrew C Simmonett, Matthew P Harrigan, Chaya D Stern, et al. 2017. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology* 13, 7 (2017), e1005659.

[20] Chongzhou Fang, Han Wang, Najmeh Nazari, Behnam Omidi, Avesta Sasan, Khaled N Khasawneh, Setareh Rafatirad, and Houman Homayoun. 2021. Reptack: Exploiting cloud schedulers to guide co-location attacks. *arXiv preprint arXiv:2110.00846* (2021).

[21] Ethan Ferguson, Adam Wilson, and Hoda Naghibijouybari. 2024. WebGPU-SPY: Finding Fingerprints in the Sandbox through GPU Cache Attacks. *arXiv preprint arXiv:2401.04349* (2024).

[22] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[23] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 195–210.

[24] Epic Games. 2019. Unreal Engine. https://www.unrealengine.com.

[25] Stefan Gast, Hannes Weissteiner, Robin Leander Schröder, and Daniel Gruss. 2025. CounterSEVeillance: Performance-Counter Attacks on AMD SEV-SNP. In *Network and Distributed System Security Symposium 2025: NDSS 2025*.

[26] Cheng Gu, Yicheng Zhang, and Nael B. Abu-Ghazaleh. 2025. I know What You Sync: Covert and Side Channel Attacks on File Systems via syncfs. In *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, Marina Blanton, William Enck, and Cristina Nita-Rotaru (Eds.). IEEE, 3636–3652. doi:10.1109/SP61157.2025.00209

[27] John K Haas. 2014. A History of the Unity Game Engine. https://api.semanticscholar.org/CorpusID:86824974.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[29] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[30] Mohammad Hossin and Md Nasir Sulaiman. 2015. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process* 5, 2 (2015), 1.

[31] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. 2020. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 385–399.

[32] Intel. 2024. Intel 64 and IA-32 Architectures Software Developer's Manual. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[33] Milan Jaroš, Lubomír Říha, Petr Strakoš, and Matěj Špet'ko. 2021. GPU accelerated path tracing of massive scenes. *ACM Transactions on Graphics (TOG)* 40, 2 (2021), 1–17.

[34] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[35] Oliver Kramer. 2013. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*. Springer, 13–23.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[37] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.

[38] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*. IEEE, 605–622.

[39] Yan Meng, Yuxia Zhan, Jiachun Li, Suguo Du, Haojin Zhu, and Xuemin Shen. 2024. De-Anonymizing Avatars in Virtual Reality: Attacks and Countermeasures. *IEEE Transactions on Mobile Computing* (2024).

[40] Frederic P Miller, Agnes F Vandome, and John McBrewster. 2009. Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance.

[41] Hoda Naghibijouybari, Khaled N Khasawneh, and Nael Abu-Ghazaleh. 2017. Constructing and characterizing covert channels on GPGPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 354–366.

[42] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered insecure: GPU side channel attacks are practical. In *Proceedings*

of the 2018 ACM SIGSAC conference on computer and communications security. 2139–2153.

[43] Vivek C Nair, Gonzalo Munilla-Garrido, and Dawn Song. 2023. Going incognito in the metaverse: Achieving theoretically optimal privacy-usability tradeoffs in VR. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–16.

[44] Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu. 2021. (Mis) managed: A novel TLB-based covert channel on GPUs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 872–885.

[45] Ravan Nazaraliyev, Yicheng Zhang, Sankha Baran Dutta, Andrés Márquez, Kevin J. Barker, and Nael B. Abu-Ghazaleh. 2025. Not so Refreshing: Attacking GPUs using RFM Rowhammer Mitigation. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, Lujo Bauer and Giancarlo Pellegrino (Eds.). USENIX Association, 5641–5660. https://www.usenix.org/conference/usenixsecurity25/presentation/nazaraliyev

[46] NVIDIA. [n. d.]. CUPTI Event API. https://docs.nvidia.com/cupti/api/group__CUPTI__EVENT__API.html. Last accessed on: 06/04/2025.

[47] NVIDIA. [n. d.]. Pascal Architecture Whitepaper. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[48] NVIDIA. 2014. NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/.

[49] NVIDIA. 2018. NVSwitch Accelerates NVIDIA DGX-2. https://developer.nvidia.com/blog/nvswitch-accelerates-nvidia-dgx2/.

[50] NVIDIA. 2019. CUPTI CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/cupti/index.html.

[51] NVIDIA. 2019. Security Bulletin: NVIDIA GPU Display Driver - February 2019. https://nvidia.custhelp.com/app/answers/detail/a_id/4772.

[52] NVIDIA. 2024. CUDA Profiler Users Guide. https://docs.nvidia.com/cuda/profiler-users-guide/.

[53] Riccardo Paccagnella, Licheng Luo, and Christopher W. Fletcher. 2021. Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael D. Bailey and Rachel Greenstadt (Eds.). USENIX Association, 645–662. https://www.usenix.org/conference/usenixsecurity21/presentation/paccagnella

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[55] Kartik Patwari, Syed Mahbub Hafiz, Han Wang, Houman Homayoun, Zubair Shafiq, and Chen-Nee Chuah. 2022. DNN model architecture fingerprinting attack on CPU-GPU edge devices. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 337–355.

[56] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*. 199–212.

[57] Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clementine Lucie Noemie Maurice, Raphael Spreitzer, and Stefan Mangard. 2018. Keydrown: Eliminating software-based keystroke timing side-channel attacks. In *Network and Distributed System Security Symposium 2018*. 15.

[58] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. 2017. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript. In *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3–7, 2017, Revised Selected Papers* (Sliema, Malta). Springer-Verlag, Berlin, Heidelberg, 247–267. doi:10.1007/978-3-319-70972-7_13

[59] Ankit Sethia and Scott Mahlke. 2014. Equalizer: Dynamic tuning of GPU resources for efficient execution. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 647–658.

[60] Mert Side, Fan Yao, and Zhenkai Zhang. 2022. 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, 270–285.

[61] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[62] Alan Smith and Vamsi Krishna Alla. 2025. AMD Instinct™ MI300X: A Generative AI Accelerator and Platform Architecture. *IEEE Micro* (2025).

[63] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[64] Blender Studio. 2023. Oti. https://studio.blender.org/characters/5d403e7aee3219164b952e2b/v2/.

[65] Blender Studio. 2023. Pinguino. https://studio.blender.org/characters/5d403ddeee3219164b952e27/v2/.

[66] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[67] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. 2021. Invisible probe: Timing attacks with pcie congestion side-channel. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 322–338.

[68] Hritvik Taneja, Jason Kim, Jie Jeff Xu, Stephan Van Schaik, Daniel Genkin, and Yuval Yarom. 2023. Hot Pixels: Frequency, Power, and Temperature Attacks on GPUs and Arm SoCs. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6275–6292.

[69] OpenMM team. [n. d.]. OpenMM Benchmark. https://openmm.org/benchmarks.

[70] Kartik Teotia, Xingang Pan, Hyeongwoo Kim, Pablo Garrido, Mohamed Elgharib, and Christian Theobalt. 2024. HQ3DAvatar: High-quality implicit 3d head avatar. *ACM Transactions on Graphics* 43, 3 (2024), 1–24.

[71] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[72] Junpeng Wan, Yanxiang Bi, Zhe Zhou, and Zhou Li. 2022. MeshUp: Stateless cache side-channel attack on CPU mesh. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1506–1524.

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

[74] Zhendong Wang, Xiaoming Zeng, Xulong Tang, Danfeng Zhang, Xing Hu, and Yang Hu. 2022. Demystifying Arch-hints for Model Extraction: An Attack in Unified Memory System. *arXiv preprint arXiv:2208.13720* (2022).

[75] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 125–137.

[76] Qiumin Xu, Hoda Naghibijouybari, Shibo Wang, Nael Abu-Ghazaleh, and Murali Annavaram. 2019. GPUGuard: mitigating contention based side and covert channel attacks on GPUs. In *Proceedings of the ACM International Conference on Supercomputing*. 497–509.

[77] Boyuan Yang, Ruirong Chen, Kai Huang, Jun Yang, and Wei Gao. 2022. Eavesdropping user credentials via GPU side channels on smartphones. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 285–299. doi:10.1145/3503222.3507757

[78] Yicheng Zhang, Ravan Nazaraliyev, Sankha Baran Dutta, Nael B. Abu-Ghazaleh, Andres Marquez, and Kevin J. Barker. 2024. Beyond the Bridge: Contention-Based Covert and Side Channel Attacks on Multi-GPU Interconnect. In *International Symposium on Secure and Private Execution Environment Design, SEED 2024, Orlando, FL, USA, May 16-17, 2024*. IEEE, 35–36. doi:10.1109/SEED61283.2024.00014

[79] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael B. Abu-Ghazaleh. 2023. It's all in your head(set): Side-channel attacks on AR/VR systems. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3979–3996. https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-yicheng

[80] Zhenkai Zhang, Tyler Allen, Fan Yao, Xing Gao, and Rong Ge. 2023. TunneLs for Bootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 960–974. doi:10.1145/3576915.3616672

[81] Zhenkai Zhang, Kunbei Cai, Yanan Guo, Fan Yao, and Xing Gao. 2024. Invalidate + Compare: A Timer-Free GPU Cache Attack Primitive. In *33rd USENIX Security Symposium (USENIX Security 24)*. 2101–2118.

[82] Zirui Neil Zhao, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. 2024. Everywhere All at Once: Co-Location Attacks on Public Cloud FaaS. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 133–149.

[83] Zirui Neil Zhao, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. 2024. 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*. 582–600.