Veiled Pathways: Investigating Covert and Side Channels within GPU Uncore

Yuanqing Miao
Penn State University
University Park, PA, USA
ypm5112@psu.edu

Yingtian Zhang
Penn State University
University Park, PA, USA
yjz5396@psu.edu

Dinghao Wu
Penn State University
University Park, PA, USA
dinghao@psu.edu

Danfeng Zhang
Duke Univeristy
Durham, NC, USA
danfeng.zhang@duke.edu

Gang Tan
Penn State University
University Park, PA, USA
gtan@psu.edu

Rui Zhang
Penn State University
University Park, PA, USA
rmz5227@psu.edu

Mahmut Taylan Kandemir
Penn State University
University Park, PA, USA
mtk2@psu.edu

***Abstract*—With the emergence of GPUs as first-class compute engines, more concentrated focus has been put into covert and side channel discovery in these architectures. However, most of the covert and side channels uncovered on GPUs to date are rooted in "GPU cores", which include computational cores, cache and core interconnects, but they do *not* consider "GPU uncore", which include non-computational engines, GPU DRAM, host-GPU links and inter-GPU links.**

In this paper, we delve into the less-explored domains of GPU uncore, unveiling four novel leakage sources for covert and side channel exploitation: (1) GPU DRAM frequency scaling; (2) NVENC utilization; (3) NVDEC utilization; (4) NVJPEG utilization. What makes these covert and side channels interesting is that they all take effect under the GPU MPS mode – which fractionalizes GPU cores and GPU memory on both desktop-scale and server-scale GPUs. Furthermore, our study reevaluates PCI-e bandwidth allocation on GPUs. Notably, we have engineered covert and side channel capable of bypassing GPU MIG isolation – a mechanism implemented by NVIDIA to physically segregate hardware resources on server-scale GPUs. Our research showcases concrete examples of these covert and side channels, highlighting their potency in breaching system security, all achieved without necessitating root privileges. This underscores the practical implications and urgency of addressing these vulnerabilities in GPU architectures.

***Index Terms*—Side channel leakage, GPU, MPS, MIG, video encoder, video decoder**

I. Introduction

As a Single-Instruction-Multiple-Threads (SIMT) processor optimized for throughput-sensitive tasks, GPUs have already demonstrated performance superiority across numerous application domains. In fact, from small mobile devices to large data centers, GPUs have proven to be both essential and irreplaceable. Originally designed for graphics processing, the parallel architecture of a GPU enables it to simultaneously process millions of pixels, resulting in vivid and life-like visual experiences. GPUs have also emerged as the backbone of modern machine learning (ML) ecosystems, particularly with the rapid expansion of ML models, algorithms, and datasets. In particular, deep learning algorithms rely heavily on the massive parallel processing capabilities of GPUs to train large neural networks and provide speedy and accurate inference results for users.

With the advent of GPUs, vulnerabilities on these powerful and widely-used compute engines require more attention, so as to prevent costly errors and faults. Among the vulnerabilities, "side channels" and "covert channels" stand out, as these unintended information channels cannot be simply mitigated by adding software patches. Such vulnerabilities root from the intrinsic design patterns in architecture – sharing different device components among different processes to maximize device utilization and throughput. Side channels and covert channels utilize the information such as time [23], [41], [64], [89], temperature [61], electromagnetism [59], [90], and acoustics [22] that are unintentionally leaked. Side channels are built by attacker to steal secrets on victim, and they usually rely on the "gadgets" in victim code that has secret-dependent footprint on microarchitecture. Contrasting with side channels, a covert channel is built by a "sender-receiver" pair to transmit information secretly across the system. A lot of efforts from both academia and industry have been put into discovering [23], [40], [41], [44], [50], [54], [57], [68], [75], [82], [83], [86], [89] and/or mitigating [24], [55], [56], [60], [63], [71], [87] side channels and covert channels on CPUs. Compared to CPUs, GPUs, particularly those manufactured by NVIDIA, do not have as many open-source documents, preventing themselves from being studied thoroughly for potential information leakage channels. However, with GPUs gaining popularity and widespread usage in different application domains, the GPU-based side channels and covert channels certainly deserve more research.

Motivated by the observations above, in this paper, we focus on GPU covert and side channels. Previous work has studied the insufficient isolation that gives rise to covert channels on shared memories [47], caches [46], [64], [65] and interconnects [20], [38] in GPUs. However, there are further areas of concern in GPUs, especially in the "uncore" part, which excludes the CUDA engine (with cache) but includes GPU DRAM, video decoding/encoding engines, JPEG image

processing engines, as well as the copy engines that transfer data via the PCIe links between the GPU and the other compute engines/memories in the system, etc.

Thus, we explore covert and side channels on the NVIDIA "GPU uncore" and discover new sources of leakage in GPU hardware. The covert and side channels can be built between two processes on the same desktop GPU or even between two isolated instances on the same server GPU with the most strict isolation guarantee on NVIDIA GPU – MIG (Multi-Instance GPU) mode [8]. Below are the leakage sources we have identified:

GPU DRAM. Researchers have explored side channels and covert channels on DRAM, which take advantage of GPU bank conflicts [46], [85] and row buffers [68]. The threats posed by these channels can be mitigated by partitioning DRAM between users or employing a closed-row policy [33], [73], [85]. While DRAM partitioning seems to be a promising solution to covert and side channel attacks, our work leverages "frequency scaling" to build a new covert channel via GPU DRAM, to challenge the partition-based mitigation.

NVENC, NVDEC and NVJPEG. To the best of our knowledge, this work is the first to explore covert and side channels on GPU video/image encoding/decoding engines. As custom hardware for media processing related tasks, NVENC, NVDEC and NVJPEG are implemented in *both* desktop GPUs and server GPUs. Through our experiments, we have surprisingly found that such encoding/decoding engines are *not* partitioned even with the MPS mode [9] on. We exploit user-mode APIs (root access is not required) from the NVML library [14] provided by NVIDIA to study the factors that affect the utilization of the encoder/decoder. Capitalizing on those factors, we next build "utilization-based" covert and side channels in NVENC, NVDEC and NVJPEG on desktop GPUs and server GPUs when operating under the MPS mode [9].

PCI-e Bus I/O. A discrete GPU is installed in the system via a PCI-e slot on a motherboard, which enables it to communicate with CPU/memory through the PCI-e bus. Previous works [78], [79] have studied how to build PCI-e bus based covert channels that enable secret communication between GPU and RDMA NIC and between GPU VMs. This work further explores how to build a PCI-e bus based covert channel between GPU instances under the MIG mode [8]. MIG is a mode in server-class GPUs which partitions GPU into multiple "GPU instances", each owning isolated computing/memory paths and engines. By our PCI-e bus based covert channel for GPU instances, we enable a secret communication channel even between the MIG instances, thus further challenging the MIG isolation guarantees [92].

Summary of Our Contributions. We explore various characteristics of NVIDIA GPU uncore and exploit them to establish new covert and side channels that can *bypass* the MPS isolation guarantee and even the MIG isolation guarantee in certain cases.

- We investigate GPU DRAM frequency scaling, including how to trigger a "frequency" increase and how to lower frequency as soon as possible. We develop a method for exploiting the GPU DRAM frequency to build covert channels, which can bypass the existing partition-based mitigation strategies.
- We study various factors that affect the "utilization rate" in GPU components such as NVENC, NVDEC, and NVJPEG. By controlling such factors, we then build covert and side channels in these hardware engines. To the best of our knowledge, this paper is the first study that explores the potential covert and side channels in NVIDIA encoder/decoder engines.
- We discover that PCI-e I/O that enables communication between GPU and CPU/main memory is dynamically shared across GPU MIG instances, which enables new covert and side channel under the GPU MIG mode.

Responsible Disclosure. We disclosed our findings to NVIDIA in February 2024, and they acknowledged receipt of the information.

II. Background

A. NVIDIA GPU Architecture

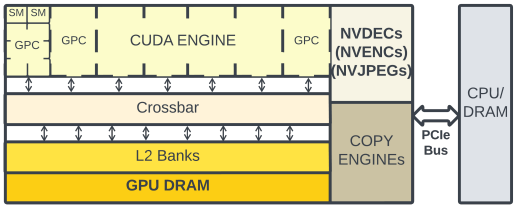


Fig. 1: A representative NVIDIA GPU architecture.

Fig. 1 shows a typical NVIDIA GPU architecture. In terms of functionality, a GPU can be divided into "compute units", "memory units", and "special engines". Compute units, like CUDA engines in NVIDIA GPUs, are composed of SMs (Streaming Multiprocessors), and each SM supports 32 threads that can execute in parallel. The CUDA engine works in an SIMT (Single Instruction, Multiple Threads) fashion, which highly benefits throughput-sensitive programs. SMs can be grouped into GPCs (Graphics Processing Clusters), and there are typically multiple GPCs in a CUDA engine.

Memory units consist of L2 cache banks and GPU DRAM. Different from register files, L1 cache and shared memory that are placed into SMs, L2 cache banks are connected to the CUDA engine via a crossbar in the GPU. The other side of L2 cache bank is lined to GPU DRAM. GPU DRAM can be of different types of memory technologies: DDR memory, usually for desktop-class GPUs, and HBM (High Bandwidth Memory), which offers significantly higher memory bandwidth and lower power consumption for server-class GPUs. We want to emphasize that the DRAM in the NVIDIA GPU does *not* share the same clock with the CUDA engine; i.e., they are in different "frequency domains".

GPUs also have special-purpose engines, including copy engines, video decoding/encoding engines [12], and JPEG processing engines [10]. A copy engine is used to move data between the GPU and the main memory in the system (not the GPU DRAM). The copy engine usually employs the DMA

(Direct Memory Access) technology without involving the CPU for every transfer operation. Video decoding/encoding engines (NVDEC/NVENC) are specific hardware units in the NVIDIA GPUs tasked with decoding/encoding video. Finally, NVJPEG is an engine employed to facilitate decoding/encoding images in the JPEG format. Note that NVJPEG hardware is currently only implemented in A100, A30, and H100 systems.

A GPU is installed on the PCI-e slot of the motherboard and thus communicates with the system via the PCI-e bus.

B. MPS and MIG Modes in NVIDIA GPUs

MPS (Multi-Process Service) [9] is a feature provided by NVIDIA GPUs that allows multiple CUDA processes to *share* a single GPU. This capability enhances resource utilization and efficiency in scenarios where multiple processes are requesting GPU concurrently. By dividing SMs (streaming multiprocessors) and memory among the concurrent processes uniformly or non-uniformly, MPS enables each CUDA process to utilize a portion of the GPU's computational capabilities. The division is *logical* – as opposed to *physical* – in that the same process is *not* necessarily pinned into the same set of SMs and memory banks during its entire execution, as long as the utilization of SMs and memory of that process does not exceed its pre-set limits. Note however that, under the MPS mode, the scheduling hardware, memory bandwidth, caches, and engines still remain shared among the MPS clients.

MIG (Multi-Instance GPU) [8] is only supported in server-class GPUs beginning with the Ampere generation. MIG is a feature that enables the *physical* partitioning of a single GPU into multiple independent "instances". Each instance has its dedicated portion of GPU resources, such as CUDA cores, L2 cache banks, memory address bus and various engines, including the copy engine and the video/image encoding/decoding engine. Note that MIG highly improves "isolation" (compared to MPS) and makes most of potential covert channels and side channels in GPUs difficult to implement.

To conclude, MIG provides more strict isolation than MPS in that MIG physically partitions SMs, memory, crossbar and engines, whereas MPS only logically divides SMs and memory by configured percentages.

C. Dynamic Voltage and Frequency Scaling

DVFS, short for dynamic voltage and frequency scaling, is a well-known technique used in computer systems to optimize power consumption and performance by adjusting the voltage and frequency of the system dynamically. DVFS is broadly used in processors to strike a balance between energy efficiency and computational performance [49], [53]. DVFS can also be used in memories and interconnects to avoid unnecessary power consumption [32]. In GPUs, there are separate clocks that control the frequency of compute units (CUDA cores), system engine and DRAM separately, and consequently, DVFS can be applied in places/locations other than main compute units.

D. NVIDIA Video and Image Processing Engines

Images and videos of high quality usually take a large storage space in their raw formats. Thus, image compression and video compression techniques (codec) have been proposed. Today's video codecs, such as HEVC [26], H.264 [81] and AV1 [2], leverages spatial and temporal similarity of consecutive frames for higher compression ratios. While there exist software libraries that enable performing encoding and decoding in CPUs, it is more efficient to *offload* these tasks to "specialized hardware units" so that the CPU is free to respond to other operations. Following the same idea, NVIDIA introduced a specific hardware for *both* image and video coding. NVJPEG [10] is the unit for JPEG-format image encoding/decoding, available in server-class GPUs like A100, A30, and H100. Starting with the Kepler series, NVIDIA GPUs are equipped with hardware NVENC for encoding videos and NVDEC for decoding videos [12], supporting processing up to 8K resolution 60fps videos at real-time speed. Although the latest NVIDIA Ada series GPUs support processing AV1 videos in both NVENC and NVDEC, AV1 codec is a very new standard and is not supported, at the time of this writing, by other hardware encoders and decoders. Hence, we will not consider it further in this study.

E. PCIe Bus between CPU/Memory and GPU

PCIe, short for Peripheral Component Interconnect Express, is a high-speed serial computer expansion bus standard. It serves as a crucial component in modern architectures, facilitating the connection and communication among various hardware components within a computer system. Specifically, CPU/memory and discrete GPU are connected via a PCIe bus, which enables high speed communication between them. GPUs also have copy engines that perform DMA operations to send/receive data to/from main memory in the system, and data is transmitted through the PCIe bus.

F. Cotenants in a GPU system

The number of cotenants in GPU systems can vary widely depending on the application and context. In high-performance computing (HPC) and data center environments, GPUs are often shared among multiple users or applications, with the number of cotenants ranging from a few to several dozens, managed by sophisticated resource allocation systems. Cloud service providers like AWS, Google Cloud, and Azure also enable multiple users to share GPU instances, leveraging some scalable infrastructure to accommodate substantial cotenancy. In virtual desktop infrastructure (VDI) setups, a single GPU can typically support 10 to 30 users [7], [18], depending on the workload intensity and GPU capabilities. For AI and machine learning applications, training clusters might share GPUs among several dozen jobs, while inference services often support multiple models concurrently [27]. Enterprise workstations using technologies like NVIDIA's vGPU can share a GPU among 4 to 16 users [11], and potentially more, depending on the workload and GPU specifications. These variations highlight the flexibility of GPU sharing across

different environments and applications and lay the basis for covert channels and side channels on GPUs.

III. Characterization

This section presents the factors that influence GPU DRAM frequency, utilization of encoder and decoder, and also PCIe bandwidth. Our finding in this section is the base for establishing corresponding covert channels and side channels.

A. DRAM Frequency

Below, we discuss our discovery regarding the GPU DRAM frequency scaling covert channels. The configuration of the GPU used in this study is described in Table I. All our experiments are performed under the MPS mode.

TABLE I: GPU configuration (Desktop).

GPU Architecture	NVIDIA GeForce RTX 3080
GPU Memory	GDDR6X 10GB
Driver Version	530.41.03
Cuda Version	12.1

With GPU GeForce RTX 3080 [5], there are 5 possible GPU DRAM frequencies:

GPU DRAM frequency (MHz)				
405	810	5001	9251	9501

When the GPU is idle, the GPU DRAM frequency sticks to 405 MHz. We refer to this frequency at 405 MHz (in the GPU DRAM) as the "idle frequency". With the MPS mode, GPU DRAM frequency will not go to 9501 MHz (the highest available frequency), but to 9251 MHz. We define 9251 MHz as "high frequency" in this paper.

To study the factors that trigger GPU DRAM frequency changes, we launch two processes at the same time. The first process (named "caller") executes CUDA kernels, and the second process (called "checker") checks the GPU DRAM frequency periodically (with a constant interval). In the checker, we use a user-mode function call from the NVML [14] library[1]: `nvmlReturn_t nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int* clock )`. By setting the second parameter to `NVML_CLOCK_MEM`, the function outputs the current GPU DRAM frequency `clock`.

Starting from an empty kernel, we have noticed that, even when the kernel is empty, the GPU DRAM frequency will increase. It will not drop back to idle frequency during the execution of the kernel. As a result, the duration of high frequency is *dependent* on the duration of kernel. However, the ending of a kernel will *not* decrease the frequency immediately. Moreover, if the interval between two kernels is too short, there will be no observed frequency drop.

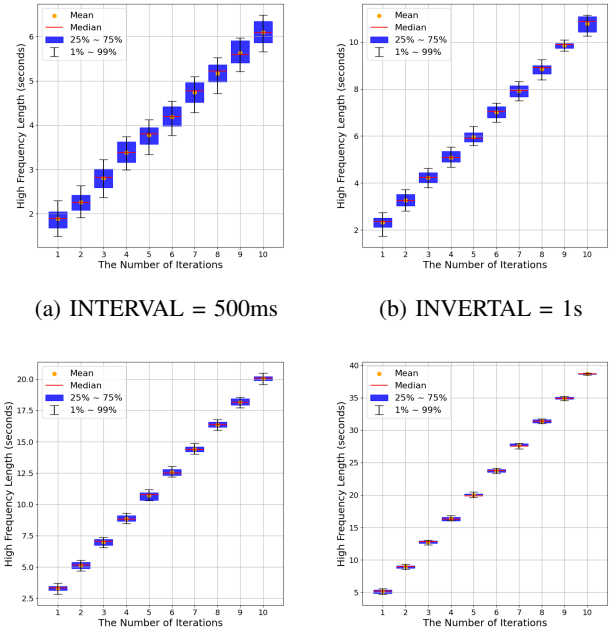
Clearly, it is important to study how to *control* frequency, in order to *encode* "secrets" in a covert channel. As the kernel execution time is not easily controllable, we design our methods based on the observation that: if the interval between kernels is too short, the frequency will not drop to the idle frequency. Thus, we set the caller as the program below:

Listing 1: Encoding ITERS into high frequency length.

```
__global__ void empty_kernel(){} //EMPTY

int main() {
    for (int i = 0; i < ITERS; ++i) {
        empty_kernel<<<1,1>>>();
        sleep(INTERVAL);
    }
}
```

We launch experiments to determine if, given a specific `INTERVAL`, the time length of high frequency is good enough to encode `ITERS`. Our results are plotted in Fig. 2. In Fig. 2a, we have `INTERVAL` set to be 500 ms and vary `ITERS` from 1 to 10. For each value of `ITERS`, we run the experiments with the caller above and the frequency checker for multiple times. We retrieve the duration of high frequency for each trial. We then calculate the mean, median, 25%~75% percentile range and 1%~99% percentile range of high frequency time in those trials. The same method is used to gather information for `INTERVAL = 1s` (Fig. 2b), `INTERVAL = 2s` (Fig. 2c) and `INTERVAL = 4s` (Fig. 2d). As the value of `INTERVAL` grows, it becomes easier to distinguish the high frequency length into different `ITERS` groups.



(a) INTERVAL = 500ms (b) INVERTAL = 1s

(c) INTERVAL = 2s (d) INTERVAL = 4s

Fig. 2: The time length of high frequency with different values of INTERVAL. Each figure shows the mean, median, 25% ~ 75% percentile, and 1% ~ 99% percentile range of high frequency duration for ITERS set from 1 to 10.

[1]NVML provides an interface for monitoring and managing various states within NVIDIA GPUs.

Takeaway-1: The duration of high frequency is controllable by launching kernels and setting a short interval in between. High frequency time length can be used to encode information. Further, the longer the interval, the lower the error rate.

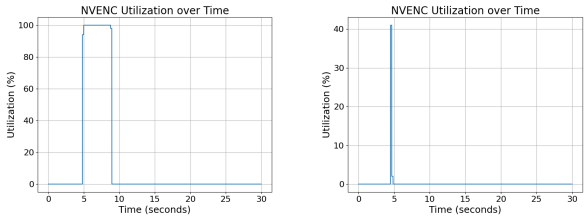
Apart from controlling the length of high frequency, it is also critical to control the frequency of the GPU DRAM to make sure that it drops to its idle frequency value. We found that the GPU DRAM frequency will *not* drop immediately when a kernel finishes, as stated earlier. However, with CUDA API `cudaDeviceReset`, the waiting time for frequency drop is shortened, that is, this API destroys all allocations and resets all states, including frequency on the current device [3]. Note that our measuring process ("checker") does not launch any kernels but only checks GPU frequency, hence, will *not* get terminated by `cudaDeviceReset`.

B. Encoder

NVIDIA GPUs have dedicated hardware support for efficient compression – NVENC. The compute-intensive video encoding task thus gets *offloaded* from the CPU to this dedicated engine in the GPU. We use `ffmpeg` to encode a video and monitor the utilization of NVENC by the user-mode API `nvmlReturn_t nvmlDeviceGetEncoderUtilization ( nvmlDevice_t device, unsigned int* utilization, unsigned int* samplingPeriodUs )` from the NVML [14] library. The NVENC utilizations of two workloads (large and small) with two standards (H.264 and HEVC) are given in Fig. 3. The details of the large and small workloads are shown in Table II.

TABLE II: "Large" and "Small" workloads.

	Large	Small
Resolution	3840 x 2160	384 x 216
Original Bit Rate	5 Mbps	300 Kbps
Target Bit Rate	1 Mbps	100 Kbps
Frame Rate	25 Fps	25 Fps



(a) large (b) small

Fig. 3: NVENC utilization with two workloads (Large and Small).

With the large workload, the H.264 and HEVC standards both bring the utilization to 100% almost immediately (Fig. 3a) and hold it there until the end of the encoding task. This result means that NVENC uses its full bandwidth to handle the tasks. With the small workload on the other hand, the NVENC utilization does not achieve 100% and the peak time is very short (Fig. 3b). The main reason for this behavior is that the small workload ends very quickly, and it does not have the chance to saturate NVENC during its processing time.

To figure out what factors influence the full utilization duration, we modify the frame rate, video length, resolution weight, and resolution height of the video separately. We collect the NVENC utilization data when encoding those modified videos. We have found that NVENC full utilization duration is *linear* with the frame rate, video length, resolution width (W), and resolution height (H).

$$Duration \propto frame\_rate \times video\_length \times W \times H \quad (1)$$

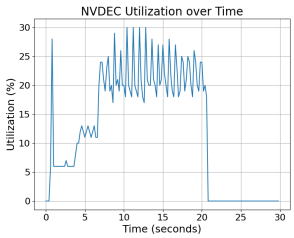
Takeaway-2: NVENC uses its full bandwidth to handle encoding tasks. However, if the workload is too small to saturate NVENC, a less than 100% utilization can be observed. The duration of NVENC full utilization is linear with frame rate, video length, resolution width, and resolution height.

We next explore how multiple processes are handled in NVENC by launching multiple encoding tasks together. In our experiments, each task is the same with the large workload in Table II, and we vary the count of tasks. When two tasks are launched together, the time duration for those two tasks both doubles, compared with only one task launched, indicating that the tasks are being processed together. As the number of tasks grows, the latency of each task slows down but remains close to others. Hence, we draw the conclusion that the encoding tasks are being processed simultaneously. There is a limit on the number of concurrent encoding sessions [13]. When this limit is reached, no more encoding tasks are accepted.

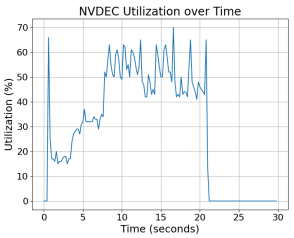
C. Decoder

NVDEC is a decoder embedded in NVIDIA GPUs, responsible for executing video decoding tasks [4]. It effectively relieves the CPU from the burden of performing resource-intensive decoding computations. We use user-mode API `nvmlReturn_t nvmlDeviceGetDecoderUtilization ( nvmlDevice_t device, unsigned int* utilization, unsigned int* samplingPeriodUs )` from the NVML [14] library to get utilization statistics for NVDEC. In the experiments, we have videos played by the VLC media player [19], which is a free and open-source cross-platform multimedia player and framework. We monitor the utilization of NVDEC.

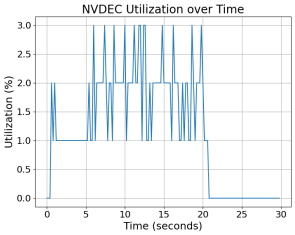
We have copies of the same video but with different resolutions, encoding standards and frame rates. We have them played, and gather NVDEC utilization. One example is illustrated in Fig. 4.



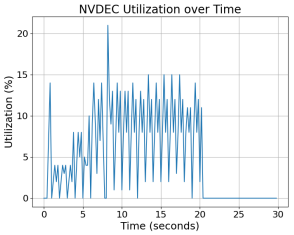
(a) 3840x2160, HEVC, 25 FPS



(b) 3840x2160, H.264, 25 FPS



(c) 384x216, HEVC, 25 FPS



(d) 3840x2160, HEVC, 10 FPS

Fig. 4: The NVDEC utilizations are different when playing videos of same content but with different encoding standard, resolution and frame rate.

Takeaway-3: Encoding standards as well as video characteristics such as resolution and frame rate collectively contribute to the differences in NVDEC utilization.

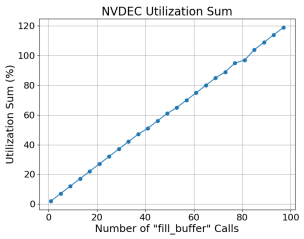
Apart from playing videos to trigger NVDEC into its usage, we can also use the supported video processing APIs [1] to make NVDEC busy. `TorchAudio` provides the class `torchaudio.io.StreamReader` to fetch and decode audio/video streams chunk by chunk. Method `fill_buffer` processes packets and adds chunks to the buffers in hardware. If the buffers are full, the pending frames are flushed. Method `pop_chunks` pops one chunk from all the buffers. We use the code in Listing 2 to study the relationship between the NVDEC utilization and the times that a stream buffer is filled (`CHUNK_NUM` in the code). To avoid frames being flushed, we add a call to `pop_chunks` after each call to `fill_buffer`.

Over time, the NVDEC utilization can be non-zero for a while. We define "NVDEC usage" as the sum of all non-zero utilization. In the experiments, we find that the NVDEC usage is *linear* with `CHUNK_NUM` (Fig. 5a).

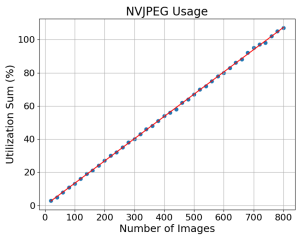
Listing 2: Using "torchaudio" to decode video.

```
...
s = StreamReader(src) # src is mp4 video
s.add_video_stream()
for i in range(0, CHUNK_NUM):
    s.fill_buffer()
    (video,) = s.pop_chunks()
```

Takeaway-4: The NVDEC usage over time correlates with the number of times that a stream buffer is filled.



(a) NVDEC usage with varied "CHUNK_NUM".



(b) The NVJPEG usage with varied number of images.

Fig. 5: Fine-control of NVDEC and NVJPEG usage.

D. NVJPEG

In addition to the engines that provide hardware support for video processing, NVIDIA also accommodates specific hardware for JPEG processing – NVJPEG, in its server-class GPUs. We performed experiments in an NVIDIA A100 [27] with the settings in Table III, to study the utilization of this NVJPEG engine.

TABLE III: GPU configuration (Server)

GPU Architecture	NVIDIA A100-SXM4-80GB
Driver Version	535.129.03
Cuda Version	12.2

We use the user-mode API `nvmlReturn_t nvmlDeviceGetJpgUtilization ( nvmlDevice_t device, unsigned int* utilization, unsigned int* samplingPeriodUs )` from the NVML [14] library to collect the utilization data for NVJPEG over time. Similar to the NVDEC usage, we also check NVJPEG every 200 ms and compute the overall usage by summing up non-zero utilization values. We vary the number of images being processed by NVJPEG (the images were identical), and gather the usage statistics of NVJPEG. Fig. 5b reveals that the NVJPEG usage is *linear* with the number of images being processed.

Takeaway-5: The NVJPEG usage is linear with the number of images processed by NVJPEG.

E. PCI-e Bandwidth

As stated earlier, with MIG (Multi-Instance GPU), each instance's processors have separate and isolated paths through the entire memory system. The on-chip crossbar ports, L2 cache banks, memory controllers, and DRAM address buses are all uniquely assigned to individual instances [8]. However, the PCI-e bus that connects the GPU and CPU/memory is not isolated among GPU instances.

To confirm the hypothesis above, we enable the MIG mode on the GPU with the setting given in Table III and have the GPU partitioned into 7 `1g.10gb` instances. In the following experiment, we copy data from CPU to GPU instance 0. Meanwhile, we have some of the other GPU

instances executing the same copy tasks, and time the latency for data transfer on GPU instance 0. Fig. 6 plots the latency in MIG instance 0 with the total number of working instances (having the copy task) varying from 1 to 7. The figure demonstrates that the MIG mode does not statically partition the PCI-e bandwidth among the MIG instances. As a result, the latency in MIG instance 0 is largely *linear* to the number of working instances.

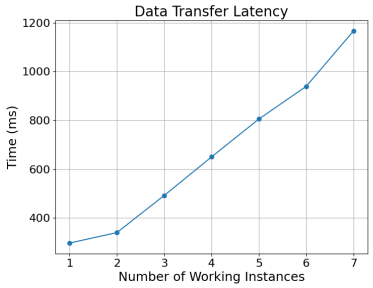


Fig. 6: Latency for data transfer from CPU to GPU.

Takeaway-6: PCI-e bandwidth is not partitioned and statically assigned to MIG instances. Consequently, the PCI-e traffic of one MIG instance can interfere with that of another.

IV. Covert Channels

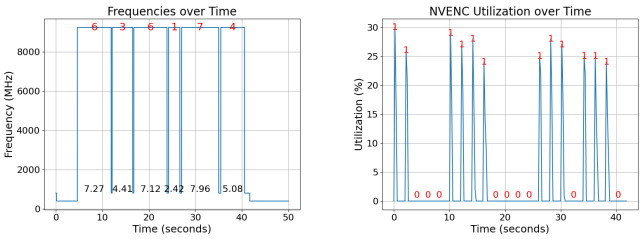
We now explain how to leverage the findings detailed in Section III to build different covert channels in NVIDIA GPUs.

Threat Models: (1) For covert channels based on GDF, NVENC, NVDEC and NVJPEG, sender and receiver are processes on the same GPU that implements the corresponding function (GPU DRAM frequency scaling) or hardware (decoder and encoder). The GPU is in the MPS mode. (2) For covert channels based on PCIe bus, sender and receiver are two processes on the same GPU in the MIG mode. Sender and receiver are in different GPU MIG instances. Root access is not required in both the cases.

A. GDF-Based Covert Channel

The GPU DRAM frequency can be used as a signal to convey information between the sender and the receiver. The advantage of this frequency-based covert channel is that it can *bypass* any partitioning-based (time and space) mitigation.

To build a covert channel by the duration of high frequency, the sender controls the duration by the code given in Listing 1. In the following example, the sender tries to send the word "cat" in ASCII format. The sender breaks the 7-bit ASCII code into 3 bits + 4 bits; hence, each letter will be transmitted by two high frequency peaks. In this example, the sender sets `INTERVAL = 1 second` in Listing 1. The sender is then ready to send "cat" by setting `ITERS` to "6, 3, 6, 1, 7, 4" in sequence and adding the `cudaDeviceReset` call between every two numbers to reset the frequency. The



(a) GDF: send "cat" by high frequency duration.

(b) NVENC: send "can" in ASCII format.

Fig. 7: Examples of GPU DRAM frequency and NVENC covert channels.

receiver then observes the GPU DRAM frequency shown in Fig. 7a. Comparing the duration of each peak with the y values in Fig. 2b, the receiver then decodes the high frequency duration into `ITERS`, which is the information transmitted by the sender. The bit rate under this attack is 0.5 bit/s.

B. Encoder-Based Covert Channel

In our experiments, we use small NVENC task with specification in in Table II to transmit information. The small NVENC task is launched when it is a bit 1, otherwise simply waits (sleeps) for a fixed period of time (2 seconds in our example). Fig. 7b shows the NVENC utilization observed by the receiver, when the sender is sending ASCII representation of "cat". The bit rate under this attack is 0.5 bit/s.

C. Decoder-Based Covert Channel

In this case, sender can encode information either to 1 and 0 by non-zero and zero utilization separately, or encode information into the duration of non-zero utilization. Apart from the above mentioned methods, we can also fine-tune the NVDEC utilization by supported video processing APIs, and use the NVDEC usage to encode information. From Fig. 5a, we obtain the linear function $y = 1.21x + 1.47$ between the NVDEC usage (y) and `CHUNK_NUM` (x), with our settings.

In our example, the sender decides to send the sequence of (6,1,6,14,6,4). The sender encodes the `value` of each part into `CHUNK_NUM` (Listing 2). To improve accuracy, we set `CHUNK_NUM` $=$ `value` $\times 5$.

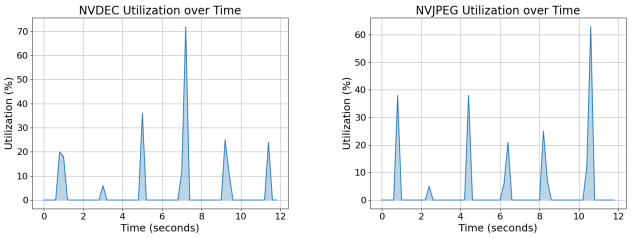
The receiver checks the NVDEC utilization every 200 ms. Fig. 8a shows the NVDEC utilization observed by the receiver. The receiver then calculates the NVDEC usage by summing up the utilization for each peak. Table IV shows the corresponding x and the closest integer to $x/5$ when setting the NVDEC usage for each peak as y, with the function $y = 1.21x + 1.47$. As a result, receiver decodes the signal of NVDEC usage into "and". The bit rate under this attack is 2 bit/s.

D. NVJPEG-Based Covert Channel

To build a covert channel via NVJPEG, we control "size of workload", i.e., the number of images being processed. We get the function of $y = 0.13x - 0.02$ from Fig. 5b by linear regression. Here, y represents the NVJPEG usage (summing

TABLE IV: x and closest integer to $x/5$ with function $y = 1.21x + 1.47$ and y.

Peak no.	1	2	3	4	5	6
y	38	6	36	84	36	24
$\lfloor x/5 \rceil$	6	1	6	14	6	4



(a) NVDEC usage. (b) NVJPEG usage.

Fig. 8: Examples of usage-based covert channel in NVDEC and NVJPEG.

up the utilization within each peak duration; the utilization is checked every 200 ms) and x represents the number of images.

We use the same method in Section IV-C to send (7,1,7,5,6,14) and gather NVJPEG usage. For lower error rate, sender processes `value` $\times$ 40 images for each `value`. The results are plotted in Fig. 8b. The bit rate under this attack is 2 bit/s.

E. PCIe-based Covert Channel

The PCIe-based covert channel can be used to convey message even under the MIG mode, which physically partitions the compute engines and memory system into multiple instances. Section III-E demonstrates that even when each instance has its own copy engine, the PCIe bandwidth between CPU and GPU is *not* partitioned, thus giving rise to a covert channel. The receiver can use the user-mode API `nvmlReturn_t nvmlDeviceGetPcieThroughput ( nvmlDevice_t device, nvmlPcieUtilCounter_t counter, unsigned int* value )` to check the PCIe throughput, like other covert channels discussed above. This function queries a byte counter over a 20ms internal and retrieves the PCIe throughput over that interval. However, this function cannot provide a very high covert channel bandwidth due to the relatively low checking frequency. As a result, we employ an approach similar to that described in [78] to build our covert channel. To transmit '1', sender copies a fixed block (S bytes) of data from host to device. To transmit '0', sender executes K nop operations. In the same time, receiver keeps copying a fixed block (R bytes) of data from host to device and measuring the latency. Fig. 9 shows an example in which the receiver decodes the latency into sequence of 0-1.

We gather the error rates with different S/R/K settings, shown in Table V.

V. Side Channels

In this section, we demonstrate side channel attacks exploiting NVDEC and NVJPEG that are feasible under the MPS mode. We also demonstrate a side channel attack that breaks MIG isolation, by exploiting PCI-e sharing. These attacks do not require root access.

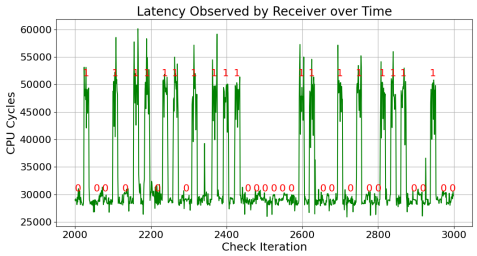


Fig. 9: Example: Latency of the receiver when the sender is sending a 0-1 sequence.

TABLE V: Error rates.

S-R-K	Bandwidth	Error Rate
64MB-1MB-1e7	93bps	0.022
8MB-64KB-6e5	1.5kbps	0.031
2MB-16KB-2e5	6.8kbps	0.033

A. Video-Contained Website Fingerprints

Putting videos (featured embedded videos and ads videos) on websites is becoming increasingly popular [17]. Videos are transmitted in an encoded format, and hence they require decoding by the user-end.

Leakage Source. Fig. 10 shows the parser-decoder-renderer-display pipeline [4] that efficiently processes video playback across different hardware components. The lightweight CPU-based parser extracts the frame data and metadata from the encoded bitstream. NVDEC then handles the decoder work, converting compressed video streams into raw frames. The renderer work is managed by the Streaming Multiprocessors (SMs) on the GPU, performing tasks such as scaling, color correction, overlay, and post-processing to prepare the frames for display. Finally, the display stage sends the prepared frames to the screen.

As shown in Section III-C, the utilization, over time, of the NVIDIA hardware decoder NVDEC is dependent on multiple video characteristics such as "resolution" and "frame rate". Therefore, videos on websites provide an additional avenue for website "fingerprints", by the NVDEC utilization trace. Because NVDEC operates independently from SM, it can perform video decoding tasks concurrently with other GPU tasks, such as rendering, without directly competing for the same compute resources. Although there is indirect competition for resources such as GPU DRAM and PCIe bandwidth, their usage does not approach their limits during our experiments below. This is primarily because the website-autoplayed videos typically have low resolution and low frame rate. Therefore, we can exclude the possibility that the rendering stage is influencing our results.

In this attack, the victim browses a website containing auto-played videos (ads, theme videos etc.), and the attacker aims

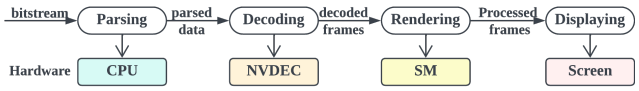


Fig. 10: Video parser-decoder-renderer-display pipeline.

to determine which website victim is browsing by analyzing the trace of the NVDEC utilization.

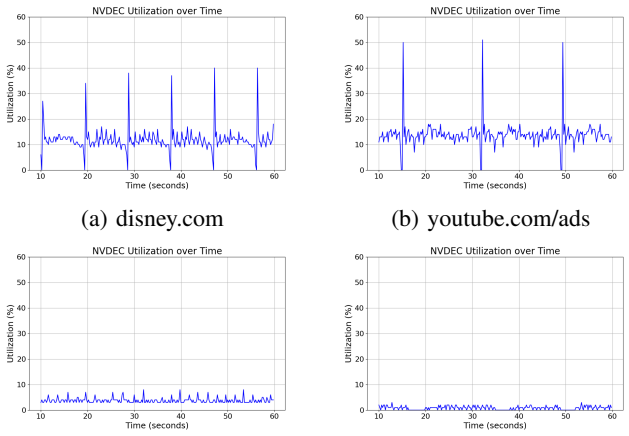
Attack Model. Website fingerprint attacks can be classified into two categories, namely, web-based and app-based [30]. A web-based attack is triggered when the victim clicks on a malicious page. Following this, a malicious code (e.g., JavaScript) gets executed in the sandbox environment of the browser. Note that such web-based attack is limited by the permissions assigned to JavaScript. An app-based attack, on the other hand, can use a malicious program or application that co-exists with the victim in the same system. In contrast to a web-based attacker, an app-based attacker is not confined to using only JavaScript. They have access to the operating system's API, enabling them to gather more traces. In this work, we consider app-based attackers who have no root access; this is similar to the attackers modeled in the previous works [35], [43], [44], [65], [80]. The attacker uses the user-mode API to gather the trace. Note that this attack is feasible under the GPU MPS mode.

Method and Result. A method similar to the existing work [44], [83] on fingerprinting can be employed on video-autoplayed website. Firstly, after gathering the traces on the NVDEC utilization, the attacker calls the same API from the NVML library to check the utilization for a set of candidate websites. Secondly, with the traces and their labels (website urls), the attacker trains an RNN classifier. Fig. 11 shows four examples of the collected traces. We use Google Chrome as our browser, and the experiments are performed with Windows 11. We gather 600 traces from 40 popular [15] video-containing websites, and train with ATT-BLSTM (Attention-based Bidirectional Long Short-Term Memory [94]), which has also been employed by the previous time-series-based fingerprint attacks [44], [83]. By leveraging the attention mechanism and bidirectional LSTM layers, the ATT-BLSTM model can effectively capture complex temporal dependencies and provide accurate predictions or detections in time series analysis tasks. Each trace is 60 seconds long, with a utilization check every 200 ms. The entire data gathering process takes 10 hours, while the model training takes approximately 3 minutes. The classification accuracy is 89.17%, with the hyper-parameters shown in Table VI.

TABLE VI: Hyper-parameters of ATT-BLSTM for website fingerprinting.

input length	hidden units	dropout rate	batch size
300	64	0.2	16

Combined with other work. By adding a new leakage resource into the attacking vectors, our work makes fingerprinting attacks more concerning. Through ensemble multi-modal learning, which combines the trained models together, we improve the accuracy of the website fingerprinting attack.



(a) disney.com (b) youtube.com/ads

(c) cnn.com (d) foxnews.com

Fig. 11: The NVDEC utilization traces captured while the victim is accessing different websites.

We combine the work presented in [65] with our fingerprinting attack. In this case, for each website, we gather two kinds of traces: one is the memory allocation size, and the other is the NVDEC utilization. We gather traces from the same 40 websites. We visit each website for 15 times and gather a 60-second-trace for each visit. We apply the RandomForest model to the 600 traces of memory allocation size and get 93.0% classification accuracy, which is slightly higher than the figure reported in the original paper [65], mainly owing to our smaller websites set. We combine the results of two separately trained models – RandomForest (memory allocation size) and ATT-BLSTM (NVDEC utilization) – into feature vectors, and train a new model (a meta-learner) using these combined features to improve performance, as illustrated in Fig. 12. Our meta-learner improves the accuracy to 98.3%.

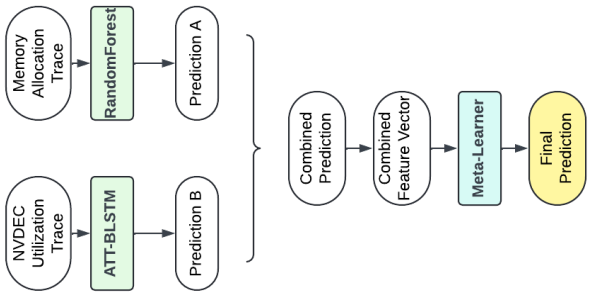


Fig. 12: The framework of our method that combine memory allocation trace and NVDEC utilization trace together to improve accuracy.

Comparison. Table VII gives a comprehensive comparison of popular architecture and micro-architecture based website

TABLE VII: Comparison of (micro)architecture-based fingerprinting attacks.

	Processor	Leakage Source	Attacker Style	# of Websites	Accuracy (%)	Sampling Rate (Hz)
Oren et al. [66]	CPU	LLC Occupancy	Web-based	8	88.6	500
Shusterman et al. [76]	CPU	LLC Occupancy	Web-based	100	87.5	345
Shusterman et al. [77]	CPU	LLC Occupancy	Web-based	100	80.0	500
Cook et al. [29]	CPU	Interrupts	Web-based	100	92.5 [1]	10000
Gulmezoglu et al. [43]	CPU	Performance counters	App-based	40	86.30	10000
Dipta et al. [35]	CPU	Core Frequency Scaling	App-based	100	97.6	100
Guo et al. [44]	CPU	Uncore Frequency Scaling	App-based	100	82.18	333
Ferguson et al. [39]	iGPU	LLC Occupancy	Web-based	100	90.6	50
Taneja et al. [80]	iGPU	Core Frequency	App-based	100	27	10
Naghibijouybari et al. [65]	GPU	GPU Memory Allocation	App-based	200	90.04	16000
NVDEC-based	GPU	NVDEC Utilization	App-based	40	**89.17**	**5**
Combined	GPU	Mem Alloc + NVDEC Util.	App-based	40	**98.33**	16000

[1] Tested on Windows with Chrome

fingerprints attacks. Our NVDEC-based attack exploits a brand new leakage source and proves itself to be among the most accurate technique. In fact, our method of combining GPU memory allocation and NVDEC utilization outperforms all the other attacks tested in accuracy.

We also compare the sampling rates used to gather fingerprinting traces in those works. Mitigation techniques that limit sampling rates, such as decreasing the resolution in performance counters and timers [60], or limit the rate an application can call relevant APIs [65], prevent attackers from achieving high classification accuracy in their models. For example, reducing timer resolution to 100 ms can decrease accuracy by nearly 50% in some attacks [29], [77]. Our NVDEC-based work maintains its performance even at a very low sample rate of 5 Hz, proving robust against such mitigations, compared to other approaches.

Variation of fingerprinting attack. In addition to identifying which website the victim is browsing, the attacker can also gain information such as which part of the website victim is interested in. This is possible because, normally, websites do not auto-play all videos at the same time. Only when the video is displaying on the screen, transmission and decoding will take place. In the following example (shown in Fig. 13), a user is browsing linkedin.com/feed, which is a mixture of text-only posts and video posts. By analyzing the trace of the NVDEC utilization, the attacker can figure out that the user has spent more time on the text content T3 between the second and the third videos, and the user is more interested in that text (by comparison, little time is spent on the text content T2 between the first video and the second video.)

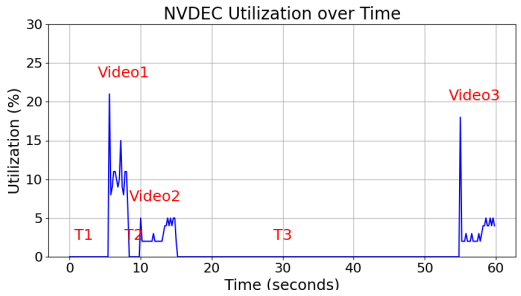


Fig. 13: The NVDEC utilization trace when a user is browsing linkedin.com/feed. This user is interested in the text content T3 between the second video and the third video on this website.

B. Model Training

The process of training machine learning models often involves handling a large number of images. While NVJPEG speeds up image processing, it also *exposes* models to side-channel attacks. In the following examples, we show how an attacker can learn secrets such as "model type" and "epoch time" of the training models in victim's process. It is to be noted that the attacker is *not* required to have kernels scheduled and launched onto the same GPU with victim, *nor* he needs root access to the system, as long as the attacker is able to check NVJPEG utilization over time by the user-mode API provided by NVIDIA. The attacker, by checking the user-mode API on a fixed frequency, shown in Section III-D, obtains the NVJPEG utilization stats over time. These attacks are feasible under the MPS mode of NVIDIA GPUs.

Model Identification: In this case, the victim is training a model on GPU. There is a set of candidate models for a specific kind of task, e.g., the ResNet family [45] for image classification. The attacker aims to discover which model is being trained. By checking the NVJPEG utilization, the attacking goal can be achieved. It is known that a training step contains three parts: i) reading in a batch of images, ii) forward propagation, and iii) back propagation. The first part uses NVJPEG to decode images in the JPEG format while the second and third parts do not. When more computation is spent over input data, the images are fetched and decoded at a lower frequency. In this case, the NVJPEG utilization rate over time on a fixed dataset is *dependent* on the time taken by the forward and back propagations, which is determined by the complexity of the models (number of layers).

We use DALI [6], NVIDIA's data loading library, to implement the pipeline of image decoding and training computations. Fig. 14 plots the NVJPEG utilization rate when using AlexNet [51], ResNet18, ResNet34 and ResNet50, to train the same subset of ImageNet [31] dataset. The more complexity (more layers) a model has, the lower the NVJPEG utilization is. As a result, by observing the NVJPEG

utilization, the attacker can gain information about which model the victim is training on the GPU.

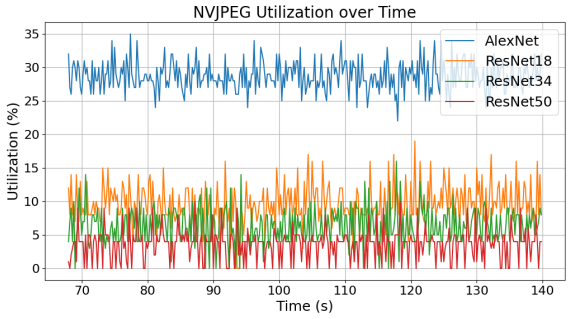


Fig. 14: The NVJPEG utilization with different models on ImageNet.

Epoch Time: The attacker can also estimate the time spent on each epoch of training by checking the NVJPEG utilization, which can be further used to guess the model [36]. Here, the behaviour of the victim is slightly different from the model identification attack. More specifically, in this case, the victim evaluates the current trained weights on a validation set after each training step. It is common in model training to add such an evaluation step after each epoch to remember the best prediction and checkpoint. The important point is that the NVJPEG utilization *differs* during training and evaluation; this is because the evaluation step does *not* include back propagation. With less computation spent on the images, the images are fetched and decoded at a higher frequency, thus resulting in a higher NVJPEG utilization. Therefore, the NVJPEG utilization is higher during evaluation (compared to training).

Fig. 15 shows an example of evaluation step after each training epoch with ResNet34 on a subset of ImageNet. The peaks and bottoms in this plot show the NVJPEG utilization of evaluation and training separately. Clearly, the attacker can obtain the epoch time by retrieving the "time interval" between two peaks (marked using red points).

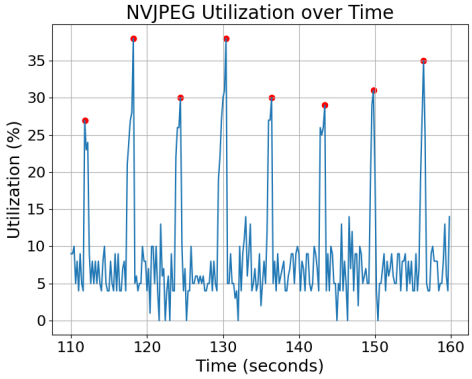


Fig. 15: The NVJPEG utilization with ResNet34 evaluation on validation set after each epoch.

C. Model Inference

With the widespread use of large language models (LLMs), it is becoming increasingly important to develop techniques to reduce memory footprints, as the cumulative weights of an LLM normally cannot fit entirely into the GPU memory. Among the memory management techniques, "weight offloading" is particularly useful. Weight offloading moves parts of the model weights between GPU and CPU memory *dynamically* during inference [21], [48], [74], [93], based on the current layer being processed.

Below, we illustrate how PCI-e leakage enables model identification attacks under MIG settings, which is to learn what language model the victim is running in an "isolated" MIG instance. In this setting we have the victim running a language model inference task in an MIG instance with offloading of the model weights. In our example, the victim uses the Zero-Inference [21] library, which is developed by DeepSpeed, supports weights offloading, and enables the inference of models with up to trillions of parameters on a single or multiple GPUs. The attacker has a set of candidate answers. Table VIII shows popular language models, which serve as the candidates in our example attack. The models differ in the sizes of their weights and how these weights are reused and kept in the GPU memory, which result in different patterns of PCIe traffic. The attacker is on the same GPU but on a different MIG instance, probing PCIe traffic patterns.

As mentioned in Section IV-E, there are two methods for PCIe traffic probing. We exploit the API-style probing by `nvmlDeviceGetPCieThroughput` to get the traffic pattern from host to GPU, with a checking frequency of 20ms. We run each inference model in Table VIII for 60 times and gather 720 traces in total. We then feed these traces into an ATT-BLSTM model, with hyperparameters being the same as those in Table VI. With that, we achieve a classification accuracy of 93.75%. Thus, by gathering the PCI-e traffic trace, the attacker can guess which model the victim is using with high precision.

TABLE VIII: Descriptions of the models.

Model	Description
BERT-base	General-purpose NLP model for various tasks [34].
BERT-large	
RoBERTa-base	Enhanced BERT for improved NLP performance [58].
RoBERTa-large	
DistilBERT	Efficient BERT with faster performance [72].
GPT2	Generative model for text generation [69].
T5-small	Versatile text-to-text transformer model [70].
ALBERT-base	Lightweight BERT for efficient processing [52].
ALBERT-large	
ELECTRA-small	Model focusing on efficient pre-training for token tasks tasks [28].
ELECTRA-base	
XLNet	Model using permutation-based training [88].

VI. Discussion

In this section, we compare our newly-identified covert channels with the related work in this domain, explain the

general applicability of our covert and side channels, and discuss the potential mitigation strategies for our newly-discovered channels.

A. Comparison of Covert Channels

Our *GPU DRAM frequency-based covert channel* leverages frequency scaling in GPU DRAM. A recent work of frequency-based information leakage on GPUs [80] studies the factors that cause frequency modulations in the GPU CUDA engine. However, the contributors including Hamming distance and Hamming weight do not take effect in NVIDIA GPUs. Also, as pointed out in [80], as the changes in frequency are small, sampling durations get longer, thus limiting their leakage rate to 0.1 bit per second. Compared to that work, we i) focus on GPU DRAM frequency, as opposed to GPU CUDA engine frequency, ii) identify the factors that can trigger GPU DRAM frequency modulations, and iii) develop corresponding strategies to fine-tune the GPU DRAM frequency, e.g., by inserting specific instructions to drop the high frequency. Apart from encoding information into frequency peaks and valleys, we also manage to encode information by the duration of high GPU DRAM frequency. As a result, we improve the bandwidth of the frequency-based information leakage channels from 0.1 bit/s to 0.5 bit/s, with high accuracy.

Our *NVENC, NVDEC and NVJPEG based covert channels* exploit the fact that there is no isolation among processes in video and image encoding/decoding engines, even when employing the MPS mode. To the best of our knowledge, ours is the *first work* that develops covert channels in such special-purpose engines. In this context, we have developed a new method that can encode information into engine usage. With the method of encoding information into duration (see the example in Section IV-B), we achieve a covert channel bandwidth of 0.5 bit/s. Further, with the method of encoding information into usage (see the example in Section IV-C and Section IV-D), the bandwidth of NVDEC- and NVJPEG-based covert channel is around 2 bits/s.

Our *PCI-e based covert channel* uses PCI-e congestion to break MIG isolation. The previous work [78], [79] studies how to utilize PCI-e congestion to build secret channels among installed devices and even GPU non-MIG VMs. In contrast, our work is the *first one* to apply PCI-e bus congestion toward the goal of establishing a covert channel between GPU instances under the MIG mode. We managed to build a highly accurate covert channel with a bandwidth with over 6kbps. This covert channel further challenges the isolation guarantees supposed to be provided by the GPU MIG mode, along with the work [92].

Table IX gives the comparison of GPU uncore covert channels, with their applicability. Here, we list the uncore hardware in GPU, excluding compute resources (core units). Although side channels and covert channels on other GPU uncore components (e.g., memory controller and crossbar) have not been thoroughly researched yet, the research methods and results on corresponding CPU uncore hardware [67], [85], can potentially be applied to them as well. Instead, in our work, we more focused on GPU-specific uncore components.

TABLE IX: Comparison of discrete GPU uncore covert and side channels, with our contributions highlighted in bold font.

Hardware	Leakage Source	Applicability
Inter-GPU links	NVLink bandwidth [91]	multi-GPU sys
	remote L2 cache [38]	
GPU DRAM	bank conflicts [46]	non-MIG
	frequency scaling	**DVFS supported**
En/De-coders	**NVENC utilization**	**NV desktop**
	NVDEC utilization	**NV desktop+server**
	NVJPEG utilization	**NV server**
Host-GPU links	PCI-e bandwidth	GPU - NIC [79]
		non-MIG VMs [78]
		MIG instances
Mem Ctr [85]*	req queue + scheduler	non-MIG
LLC	cache conflicts [37]	non-MIG
Crossbar [67]*	port contention	non-MIG

* work on CPU uncore, but can be applied to GPU uncore

B. Generality

Our GPU DRAM frequency based covert channel relies on frequency scaling. We want to emphasize that the GPU DRAM frequency scaling is implemented in almost all recent desktop-scale GPUs since the Fermi generation (launched in 2010) [25], [62], [84], as well as in some recent AMD desktop GPUs [42].

The GPUs equipped with media engines are vulnerable to the corresponding engine based covert and side channels discussed in this paper. Most of the recent desktop-scale and server-scale NVIDIA GPUs have NVENC implementation, as well as NVDEC [16]. NVJPEG is newly supported in NVIDIA A100, A30 and H100 [10].

Our PCI-e based covert channel is dependent on the PCI-e bus between GPU and CPU/memory. To the best of our knowledge, it is applicable to all discrete GPUs in the market. Furthermore, this PCI-e covert channel is expected to draw more attention in the context of NVIDIA GPUs that support the MIG mode, as it breaks the most strict isolation in GPU (i.e., the isolation between the GPU MIG instances).

C. Mitigation

For mitigating the GPU DVFS-based covert channel, one of the potential solutions could be to disable DRAM frequency scaling, which would bring, unfortunately, energy and power overheads. Another solution could be to have separate DRAM and DRAM clock domain for each and every GPU partition. But, unfortunately, doing so would cause large area overheads.

To mitigate the covert and side channels targeting the NVENC, NVDEC and NVJPEG engines under the MPS mode, the designer could statically assign a part of the bandwidth of those engines to one process. However, such bandwidth partitioning can prevent the full throughput from being achieved: even if other processes are not using their

shares of bandwidth, those shares cannot be allocated to the process that is busy with encoding/decoding videos/images.

Finally, to mitigate the PCIe covert channel that breaks the isolation under the MIG mode, the bandwidth between CPU and GPU could be statically partitioned over instances. However, doing so will most likely hurt the overall throughput and latency. Dynamically shaping PCI-e traffic [33], [95] might help to strike a balance between security and performance, which is left for our future work.

Of course, the mitigation of these newly-discovered covert and side channels requires more effort, which is in our future research agenda but beyond the scope of this paper.

VII. Conclusion

In this paper, we explore covert and side channels on NVIDIA GPU uncore, and discover four new information leakage sources on NVIDIA desktop- and server-class GPUs. We develop a strategy to fine-control GPU DRAM frequency and utilize frequency up-and-down along with high frequency duration to build covert channels based on GPU DRAM frequency scaling. Also, we study the factors that affect the utilization rate in NVIDIA GPU special purpose engines: NVENC, NVDEC and NVJPEG. To the best of our knowledge, this paper is the first to identify covert and side channels on such engines. Additionally, we reevaluate PCI-e bandwidth allocation on GPU, and build new covert and side channel between GPU instances under MIG mode, thus challenging the isolation guarantee promised by the MIG mode.

Acknowledgment

We are thankful to the anonymous reviewers for their constructive feedback. This work was supported in part by the National Science Foundation under Grant 1956032.

References

[1] "Accelerated video decoding with nvdec," https://pytorch.org/audio/main/tutorials/nvdec_tutorial.html, accessed: 2024-02-08.

[2] "Av1 bitstream & decoding process specification," https://aomediacodec.github.io/av1-spec/, accessed: 2024-02-08.

[3] "Device management functions of the cuda runtime application programming interface," https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html, accessed: 2024-02-07.

[4] "Nvdec video decoder api programming guide," https://docs.nvidia.com/video-technologies/video-codec-sdk/12.1/nvdec-video-decoder-api-prog-guide/index.html, accessed: 2024-02-08.

[5] "Nvidia ampere ga102 gpu architecture," https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf, accessed: 2024-02-08.

[6] "Nvidia data loading library (dali)," https://developer.nvidia.com/dali, accessed: 2024-04-12.

[7] "Nvidia grid vpc sizing guide," https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/solutions/resources/documents1/Application-Sizing-Guide-NVIDIA-GRID-Virtual-PC.pdf, accessed: 2024-06-24.

[8] "Nvidia multi-instance gpu user guide," https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html, accessed: 2024-02-07.

[9] "Nvidia multi-process service user guide," https://docs.nvidia.com/deploy/mps/index.html, accessed: 2024-02-07.

[10] "Nvidia nvjpeg," https://docs.nvidia.com/cuda/nvjpeg/index.html, accessed: 2024-02-07.

[11] "Nvidia rtx virtual workstation," https://images.nvidia.com/content/Solutions/data-center/sizing-guide-nvidia-rtx-virtual-workstation.pdf, accessed: 2024-06-24.

[12] "Nvidia video codec," https://developer.nvidia.com/video-codec-sdk, accessed: 2024-02-07.

[13] "Nvidia video encode and decode gpu support matrix," https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new#Encoder, accessed: 2024-02-07.

[14] "Nvml api," https://docs.nvidia.com/deploy/nvml-api/index.html, accessed: 2024-02-07.

[15] "Top websites ranking," https://www.similarweb.com/top-websites/, accessed: 2024-04-17.

[16] "Video encode and decode gpu support matrix," https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new, accessed: 2024-02-08.

[17] "Video marketing statistics 2024," https://www.wyzowl.com/video-marketing-statistics/, accessed: 2024-04-13.

[18] "Virtual desktop infrastructure (vdi) performance on intel® data center gpu flex series," https://www.intel.de/content/dam/www/central-libraries/us/en/documents/2023-08/flexseriesvdiperfwhitepaper-final.pdf, accessed: 2024-06-24.

[19] "Vlc media player," https://www.videolan.org/, accessed: 2024-02-08.

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

[21] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[22] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, C. Sporleder *et al.*, "Acoustic {Side-Channel} attacks on printers," in *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[23] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*. Springer, 2006, pp. 201–215.

[24] P. Borrello, D. C. D'Elia, L. Querzoni, and C. Giuffrida, "Constantine: Automatic side-channel resistance using efficient control and data flow linearization," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 715–733.

[25] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A simple model for portable and fast prediction of execution time and power consumption of gpu kernels," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–25, 2020.

[26] B. Bross, "High efficiency video coding (hevc) text specification draft6," in *JCTVC 7th Meeting: Geneva, CH, 21-30 Nov*, 2011.

[27] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[28] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[29] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 204–217.

[30] P. Cronin, X. Gao, H. Wang, and C. Cotton, "An exploration of arm system-level cache and gpu side channels," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 784–795.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[32] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 225–238, 2011.

[33] P. W. Deutsch, Y. Yang, T. Bourgeat, J. Drean, J. S. Emer, and M. Yan, "Dagguise: mitigating memory timing side channels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 329–343.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[35] D. R. Dipta and B. Gulmezoglu, "Df-sca: dynamic frequency side channel attacks are practical," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 841–853.

[36] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[37] 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.

[38] 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.

[39] E. Ferguson, A. Wilson, and H. Naghibijouybari, "Webgpu-spy: Finding fingerprints in the sandbox through gpu cache attacks," *arXiv preprint arXiv:2401.04349*, 2024.

[40] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with {TLB} attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 955–972.

[41] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 279–299.

[42] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "Dvfs-aware application classification to improve gpgpus energy efficiency," *Parallel Computing*, vol. 83, pp. 93–117, 2019.

[43] B. Gulmezoglu, A. Zankl, T. Eisenbarth, and B. Sunar, "Perfweb: How to violate web privacy with hardware performance events," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 2017, pp. 80–97.

[44] Y. Guo, D. Cao, X. Xin, Y. Zhang, and J. Yang, "Uncore encore: Covert channels exploiting uncore frequency scaling," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 843–855.

[45] 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.

[46] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional gpus: Software-based compute and memory bandwidth reservation for gpus," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 29–41.

[47] Z. H. Jiang, Y. Fei, and D. Kaeli, "A novel side-channel timing attack on gpus," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 167–172.

[48] K. Kamahori, Y. Gu, K. Zhu, and B. Kasikci, "Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models," *arXiv preprint arXiv:2402.07033*, 2024.

[49] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, 2008, pp. 123–134.

[50] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[52] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[53] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 487–498, 2013.

[54] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[55] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 87–101, 2015.

[56] 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.

[57] 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.

[58] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[59] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, "Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 4383–4400. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/maia

[60] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," *ACM SIGARCH computer architecture news*, vol. 40, no. 3, pp. 118–129, 2012.

[61] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *24th USENIX security symposium (USENIX security 15)*, 2015, pp. 865–880.

[62] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of gpu dvfs on energy conservation," *Digital Communications and Networks*, vol. 3, no. 2, pp. 89–100, 2017.

[63] Y. Miao, M. T. Kandemir, D. Zhang, Y. Zhang, G. Tan, and D. Wu, "Hardware support for constant-time programming," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 856–870.

[64] 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.

[65] 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.

[66] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1406–1418.

[67] 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.

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

[69] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI*, 2019.

[70] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[71] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital {Side-Channels} through obfuscated execution," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 431–446.

[72] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[73] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding information leakage in the memory controller with fixed service policies," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 89–101.

[74] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[75] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.

[76] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "{Prime+ Probe} 1,{JavaScript} 0: Overcoming browser-based {Side-Channel} defenses," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2863–2880.

[77] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 639–656. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/shusterman

[78] 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.

[79] 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.

[80] H. Taneja, J. Kim, J. J. Xu, S. van Schaik, D. Genkin, and Y. Yarom, "Hot pixels: Frequency, power, and temperature attacks on gpus and arm socs," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 6275–6292. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/taneja

[81] I. Telecom, "Advanced video coding for generic audiovisual services," *ITU-T Recommendation H. 264*, 2003.

[82] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

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

[84] Q. Wang and X. Chu, "Gpgpu performance estimation with core and memory frequency scaling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2865–2881, 2020.

[85] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 225–236.

[86] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power {Side-Channel} attacks into remote timing attacks on x86," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[87] 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.

[88] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.

[89] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[90] Z. Zhan, Z. Zhang, S. Liang, F. Yao, and X. Koutsoukos, "Graphics peeping unit: Exploiting em side-channel information of gpus to eavesdrop on your neighbors," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1440–1457.

[91] Y. Zhang, R. Nazaraliyev, S. B. Dutta, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Beyond the bridge: Contention-based covert and side channel attacks on multi-gpu interconnect," *arXiv preprint arXiv:2404.03877*, 2024.

[92] 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.

[93] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023.

[94] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, 2016, pp. 207–212.

[95] Y. Zhou, S. Wagh, P. Mittal, and D. Wentzlaff, "Camouflage: Memory traffic shaping to mitigate timing attacks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 337–348.