IOValve: Leakage-Free I/O Sandbox for Large-Scale Untrusted Data Processing

Sangho Lee
Microsoft Research
Redmond, WA, USA

Jules Drean*
Massachusetts Institute of Technology
Cambridge, MA, USA

Yue Tan*
Princeton University
Princeton, NJ, USA

Marcus Peinado
Microsoft Research
Redmond, WA, USA

Abstract

The widespread adoption of Large Language Models (LLMs) is driving the rapidly growing demand for large-scale computations like training and fine-tuning models. In many areas, the confidentiality of the underlying data is of critical importance to their corporate or government owners. However, securing data in large-scale computations is challenging. First, its demand for enormous hardware resources typically requires outsourcing (e.g., to the public cloud). Second, the large and rapidly evolving software stack used in LLM training in conjunction with a growing incidence of supply chain attacks and software vulnerabilities makes it all but impossible for data owners to establish trust in the code that processes their highly sensitive data. Confidential computing and sandboxing are promising techniques for solving these problems. However, existing sandboxes do not address covert channels which limits their ability to protect confidential data.

This paper proposes IOValve, a novel I/O sandbox for large-scale computations on confidential data. IOValve places sandbox enforcement on a programmable network device that is physically isolated from the processor hardware running the untrusted software stack. This construction allows IOValve to sidestep the multitude of side channels due to visible or hidden resource sharing. IOValve interposes on all network I/O of the sandbox and only transmits encrypted and regularized network traffic in order to prevent information leakage over the network. Our evaluation shows that IOValve has marginal performance overhead and supports real-world applications like LLM fine-tuning and batch inference, and molecular simulation.

CCS Concepts

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

Keywords

Sandbox; Covert channel; Programmable network device

*Part of this work was done while these authors were interns at Microsoft Research. Jules Drean is now at Tinfoil. Yue Tan is now at Google.

CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765121

ACM Reference Format:
Sangho Lee, Jules Drean, Yue Tan, and Marcus Peinado. 2025. IOValve: Leakage-Free I/O Sandbox for Large-Scale Untrusted Data Processing . In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765121

1 Introduction

Large-scale computations on confidential data in the cloud enable the processing of vast, sensitive datasets with power and efficiency that would be challenging and costly to achieve locally. This capability is pivotal in training and fine-tuning of Large Language Models (LLMs) and other machine learning and scientific computation tasks, where immense computational resources are necessary for handling and deriving meaningful insights from voluminous data. For example, Meta used two GPU clusters (24,576 NVIDIA H100 GPUs in each) to train their Llama 3 models [64]. Full fine-tuning of Llama 3.1 405B needs 3.25 TB of GPU memory [106], requiring more than 40 H100 GPUs.

Many potential cloud customers, such as governments, aerospace companies, financial services, or defense contractors, view the confidentiality of their sensitive data as business critical. This often makes them hesitant to fully embrace cloud computations despite their substantial benefits due to concerns over data breaches and unauthorized access [30, 51].

Confidential computing has emerged as the main tool for excluding cloud providers from the Trusted Computing Base (TCB) of their customers [13]. It places customers' sensitive data and computations into enclaves or Confidential Virtual Machines (CVMs) [8, 13, 25, 71] which are protected by processor hardware from the cloud provider's hypervisor.

However, excluding the cloud provider from the customer's TCB solves only one aspect of the data confidentiality problem. The software stack needed to support modern large-scale computations has become so complex that users effectively have no basis for trusting the code that processes their confidential data. In addition to a standard operating system (e.g., Linux) and GPU drivers and libraries (e.g., CUDA [84]), the distributed machine learning stack includes GPU management and orchestration (e.g., NCCL [89]), model parallelism tools (e.g., Alpa [113]), training libraries (e.g., PyTorch [117] and TensorFlow [114]), distributed training frameworks (e.g., Ray [9], Horovod [115]), data pipeline tools, and tools for monitoring and logging the training process. These components include rapidly evolving open and closed source software from a multitude of sources. In addition to the almost certain presence of vulnerabilities in such a large TCB [59], supply chain attacks [5, 37, 41, 110] are becoming more frequent with 633% year-over-year growth and over 88,000 individual attack instances reported in 2022 [24].

Sandboxed systems have been proposed to address this problem [3, 40, 47, 48, 74, 102, 135]. They run all code that accesses confidential data in a sandbox and control information flows out of the sandbox. This is effectively a coarse-grained form of Mandatory Access Control (MAC) [14] or Information-Flow Control (IFC) [31] and removes the code that processes the confidential data from the data owner's TCB. Several systems place the sandbox inside a TEE, thus excluding both the cloud provider and the software stack from the TCB [47, 48, 74].

In practice, the confidentiality guarantees of sandboxed systems have been severely undermined by covert channels where adversarial code inside the sandbox actively transmits information to a listener on the outside. While several systems address storage covert channels, timing covert channels are almost universally excluded from the threat model [48, 62, 121]. The intractability of timing covert channels has persisted for decades [58] and has been exacerbated by the flood of micro-architectural channels that have been discovered in recent years [20, 130]. These high-bandwidth channels effectively void all sandbox confidentiality guarantees.

The growing importance of large-scale computations presents an opportunity to rethink the sandbox boundary. The idea is to move away from the fine-grained setting in which covert channel mitigation is believed to be impossible [62, 133] and toward a much coarser boundary around entire physical hosts whose only connection to the untrusted outside world is a network wire. Eliminating covert channels from the latter appears significantly more tractable than doing so in a processor core. Large-scale computations are well suited for this purpose, as they require a set of entire machines due to a high need for processing power and because the communication pattern between these machines is generally regular.

This paper presents IOValve, a system that supports large-scale computations on confidential data while excluding both the cloud provider and the data processing software from the TCB. Importantly, IOValve includes a sandbox construction that avoids and mitigates covert channels, allowing it to protect data confidentiality. Furthermore, the IOValve sandbox can host unmodified data processing platforms such as PyTorch.

The guiding principle behind IOValve's sandbox design is to make the sandbox interface as simple as possible and to facilitate strong physical separation between the code inside the sandbox and the code enforcing the sandbox. As a result, IOValve draws the sandbox boundary at the network interface with *send* and *receive* being the only two interfaces between the sandbox and the outside world which are independent of any trusted or untrusted activities inside the sandbox.

IOValve's *sandbox monitor* interposes on all network traffic in and out of the sandbox and applies deterministic policies to it. For outbound traffic, the policies can be a combination of *encryption* (with the user's key), *block*, and *regularization* of both volume and timing. This allows IOValve to sandbox collections of entire bare metal machines on which users can run existing software. For example, a corporate cloud customer may rent one or more GPU-equipped bare metal computers in the cloud to fine-tune an LLM with business-sensitive data. Confidentiality is ensured by the sandbox monitor encrypting and regularizing (or shaping) all outbound network traffic such that untrusted code inside the sandbox cannot explicitly or implicitly exfiltrate sensitive data.

Our prototype places the sandbox monitor on a Data Processing Unit (DPU) [88] which is effectively a separate computer equipped with a network interface and connected to the main computer (i.e., the *host*) via the PCIe bus. We carefully restrict the DPU's DMA and any other PCIe bus accesses to prevent the host from congesting it. The DPU is the only networking device of the host.

The sandbox monitor running on the DPU is the user's software TCB. Remote users can provide their own monitor code and use attestation and authenticated boot to ensure that it runs on the DPU. The user can now remotely boot the host with a system image transferred through the DPU.

At this point, the user's sandbox monitor can control (i.e., block or encrypt) all network traffic generated by the host. This leaves covert channels as the only way to exfiltrate data from the host. By design, the only possible covert channel is the network wire. IOValve seeks to eliminate this network covert channel by regularizing any traffic it sends on the network wire on behalf of the host. That is, the sandbox monitor will send encrypted packets of predefined sizes with a fixed time interval irrespective of the actions and network requests of the host (with dummy data if needed). Thus, a covert channel listener (i.e., a cloud network operator) on the network can only observe periodic random bitstrings which is immune to traffic analysis. This constant or fixed shaping is a well-known concept (e.g., BuFLO [35]). We make it practical with efficient software design and careful hardware usage.

We have implemented IOValve on x86-based hosts equipped with GPUs and NVIDIA BlueField DPUs. Our evaluation includes fine-tuning of Llama 3 8B and Llama 3 70B [34] where IOValve increases their training time by 2.1%–4.1%. We also demonstrate IOValve on batch inference with Llama 3 8B whose highest token generation throughput is decreased by 1.4%. Additionally, IOValve supports training a deep learning model for molecular dynamics simulation with a time overhead of 3.4%–5.1%. It is worth noting that IOValve supports these applications without modifying them.

In summary, this paper makes the following contributions.

- We present a novel coarse-grained sandbox architecture that sidesteps traditional covert channels.
- We present a mechanism to block network-based covert channels using a programmable networking device.
- We present application case studies on LLM fine-tuning, LLM batch inference, and molecular simulation.

2 Background

In this section, we explain the background of IOValve.

2.1 Network Covert Channels

A covert channel is an undesired or unexpected channel that allows the adversary to implicitly leak secrets [92]. Network covert channels specifically refer to channels that encode secrets into network

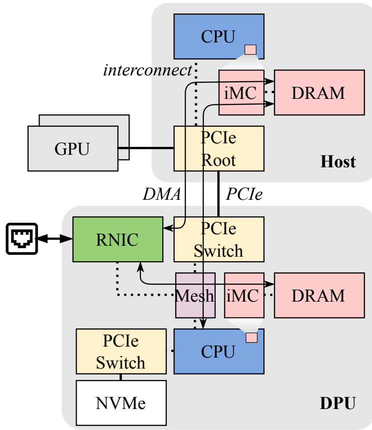


Fig. 1: Simplified host-DPU block diagram (iMC: Integrated Memory Controller, RNIC: RDMA-capable NIC).

traffic. There are two types of network covert channels: timing and storage channels.

Covert timing channel. The covert timing channel leaks secrets by modulating resource usage or network traffic such that an external party can recognize the difference based on factors such as response time or throughput. For example, the adversary can adjust the inter-packet delay of their network traffic to let the malicious network operator observe the difference. If the adversary cannot generate arbitrary network traffic due to IFC- or MAC-like strict security enforcement, they will try to slow down or block others' allowed network traffic by congesting shared hardware resources like CPU and memory, which is still visible to the outside.

Covert storage channel. The covert storage channel leaks secrets by embedding secrets into network traffic. For example, the adversary can store secrets in unused fields of IP, TCP, or UDP headers. Also, the adversary can simply adjust the packet size with padding. Mitigating the storage channel is challenging because some systems repurpose the unused fields (e.g., DDoS traceback [132]). Also, it is difficult to know what the benign packet sizes are in advance.

2.2 Data Processing Unit (DPU)

A DPU or Infrastructure Processing Unit (IPU) is a programmable hardware device that can control and manipulate the I/O (e.g., network and storage) of a computing device. Amazon Nitro Card [7], AMD Pensando [1], Intel IPU [53], Microsoft Azure Boost [78], and NVIDIA BlueField [88] are well-known DPUs. Cloud providers offload essential infrastructure security and management tasks from CPUs to DPUs (e.g., packet encryption and encapsulation and health monitoring), resulting in better and more predictable performance for cloud tenants.

Fig. 1 is a block diagram of the host and DPU with PCIe and DMA interactions based on NVIDIA BlueField-3 [88]. The DPU consists of a Remote DMA (RDMA)-capable NIC (RNIC) containing hardware accelerators (e.g., for packet encryption and encapsulation), a general-purpose (Arm) CPU, a PCIe switch, DRAM, and flash storage. The RNIC, CPU, and switch are interconnected with each other through a coherent mesh. The DPU can program its switch to selectively expose its internal components to the host. For example, the DPU can expose its RNIC to the host CPU to let them interact directly through DMA and MMIO to behave like conventional NICs or RNICs. On the other hand, the DPU can redirect any commands or data from the host to its internal CPU for various operations, including security checks and data processing, and then interact with its RNIC on behalf of the host. IOVALVE leverages the latter configuration to interpose on all I/O interactions of the host.

2.3 Collective Communication

Collective operations and communications enable distributed parallel computing like scientific computing and ML training. Distributed parallel computing typically consists of the distribution of jobs and data to each *rank* (a computation unit like a CPU core or a GPU device), individual computation by each rank, and the collection of the intermediate or final computation results of the ranks. Collective communications realize all these steps with no or minimal redundant data transfers. Three real-world applications that we run IOVALVE with, Llama 3 fine-tuning and batch inference and molecular dynamics simulation (§6.4), leverage collective communications. For example, according to our experiment, these applications use the four collective operations provided by the NVIDIA Collective Communications Library (NCCL) [89]: AllGather, AllReduce, Broadcast, and ReduceScatter. AllGather collects data from all ranks and stores them on all ranks. AllReduce collects data from all ranks, combines (or reduces) them using a specific operator, and distributes the result to all ranks. Broadcast allows one rank to distribute data to all other ranks. ReduceScatter is like AllReduce except that it scatters some portions of the result to each rank.

The operations and payload sizes of collective communications are almost uniform. Thus, the extra effort to regularize them (e.g., pad small packets, segment large packets, and adjust packet transmit intervals) is marginal. This allows us to realize covert-channel-free collective communications with moderate overhead (§6.3).

3 Model and Goal

In this section, we explain this paper's threat model and goals.

3.1 Threat Model

IOVALVE's focus is on protecting the confidentiality of the data owners' data. We exclude the availability and integrity of computation. The core tenets of our threat model are:

Data owners do not trust the software processing their data. As described in §1, the software stack for modern large-scale computations is highly complex, resulting in a high potential for exploitable vulnerabilities and supply chain attacks [5, 41, 110].

Data owners do not trust any of the cloud provider's software. This is the core promise of most work on confidential cloud computing [13, 107]. Going beyond that standard, our threat model includes all software- and hardware-based side channels which are software controllable and do not require physical access (e.g., most microarchitectural side channels [73]). Furthermore, we cover any attempt to use such side channels to build covert channels between sandboxed untrusted data processing software and the cloud software or any entity outside the cloud. Finally, the cloud provider's software-controllable network infrastructure (e.g., routers, load balancers, and switches) is also untrusted.

Data owners trust the cloud provider's hardware and its physical security. This assumption is implicit in most of the literature on confidential cloud computing based on TEEs, as a sufficiently powerful hardware attack can break any TEE security guarantees (e.g., voltage glitching [18, 23] or invasive hardware attacks such as electron microscopy [26], passive voltage contrast [142], focused ion beam editing [44], and electromagnetic pulse [118]).

However, our threat model excludes even simpler hardware attacks such as tampering with the motherboard configuration. Why is it reasonable to include all software attacks while excluding all hardware attacks? Does this combination provide meaningful protection to data owners? To answer these questions, we observe that the term 'cloud provider' is effectively a proxy for several independent actors and analyze them individually: (a) the company that controls the cloud, (b) its employees, (c) external hackers.

(a) The company: We consider large reputable corporations (e.g., Amazon, Microsoft, and Google). They have an overwhelming incentive to abide by the law and customer contracts. Violations would risk catastrophic damage to their business due to loss of customer trust, reputation damage and legal penalties. We do not consider the company itself to be an attacker.

(b) Employees: Cloud companies employ thousands to tens of thousands of software developers, administrators, and operations personnel [28]. Many of these employees are insiders according to definitions by CISA [27] and Mandiant [56]. Thousands of insider attacks have been documented in government and industry [55, 60, 82, 119]. Our threat model covers this large attack surface.

Datacenter employees might attempt hardware attacks, but this is mitigated by the small on-site workforce [94], physical security (e.g., CCTV and armed guards) [100], and their limited reach—just one facility versus the full cloud accessible to developers and admins.

(c) External hackers: Our model includes hackers who might have 'hacked' into the cloud. They may try to launch software attacks (which are covered by our threat model), but they cannot run hardware attacks (because they require physical access).

Data owners trust IOValve. The IOValve firmware and software are remotely attestable [83]. It does not receive and execute any external code and commands—it only accepts the data owner's code and security policies for I/O sandboxing during provisioning and uses them until it is reset. Since it is within the datacenter, its physical security is guaranteed by the same means. Except for crypto keys, its configuration is public.

3.2 Goal

We aim to achieve the following security goal:

> The IOValve sandbox prevents *untrusted software* which processes confidential data in the public cloud from *exfiltrating* the data to *untrusted entities*.

The *untrusted software* includes data processing software, any libraries it depends on, and the underlying system software like the operating system and the hypervisor. An *untrusted entity* is any remote entity outside the user's IOValve sandbox to which the user does not intend to disclose their confidential data. An example could be a network switch on which a rogue cloud employee may have installed traffic monitoring software. *Exfiltration* covers both overt and covert information leakage. That is, IOValve prevents the untrusted software from creating network connections to arbitrary untrusted entities and encrypts all legitimate network traffic (e.g., to the user's on-premises machines or between the IOValve sandboxes). It also prevents the untrusted software from modulating legitimate network traffic in a way that is recognizable by the untrusted entity.

4 Design

In this section, we explain the design of IOValve. IOValve ensures confidential data processing without covert channels using a programmable network device (the IOValve DPU) that controls and regularizes all I/O traffic of each computing node. We explain the configuration needed for hardware-level security (§4.1), system provisioning (§4.2), congestion avoidance (between host and DPU §4.3) and inter-node (between DPUs §4.4) communication.

4.1 Hardware and System Configuration

IOValve requires the following configurations to ensure its security. First, the IOValve DPU must be the sole network device of a computing node to interpose on all network traffic. There should be no other internal (e.g., PCIe or motherboard integrated) and external (e.g., USB) NICs plugged into the node. We consider this to be a physical attack which is out of this paper's scope. Second, the IOValve DPU must be able to ignore any administration commands from the node which is untrusted. The administration commands include DPU firmware flashing, hardware component exposure, and network configuration. Many DPUs including the BlueField DPU that IOValve relies on already support this security feature [87] to deal with node compromise. Third, the IOValve DPU must be able to (re-)provision the node. That is, the IOValve DPU can reset the node at any time and reinstall its operating system from scratch (e.g., through Preboot eXecution Environment (PXE) network boot or boot storage emulation). It is worth noting that IOValve does not require any hardware modification to ensure this re-provisioning—we allow the IOValve DPU to control the node's Baseboard Management Controller (BMC) as AWS Nitro does [7]. Fourth, the IOValve DPU must be controlled by a single tenant to avoid any potential co-tenant side-channel attacks.

4.2 Provisioning and Attestation

We describe how IOValve provisions the DPU and the host with the help of attestation. IOValve relies on bare-metal provisioning with attestation schemes for cloud nodes and Arm machines [12, 81]. Thus, instead of mentioning the details of provisioning mechanisms here, we focus on our insight to leverage them (i.e., their network boot and attestation) to realize two-level provisioning as follows.

DPU provisioning. IOValve provisions the DPU with a user-provided image once it is turned on or power cycled. To this end, we run a small Linux operating system (`initramfs`) on the DPU. This small operating system runs an HTTPS server to receive a user's request, downloads an image to a separate partition, and resets itself to boot with the new image. This user-provided image contains the DPU-side IOValve libraries.

DPU-driven host provisioning. Once the DPU has been provisioned with the user-provided DPU image, it provisions the host (or node) it is plugged into. It first receives a system image for the host from the user (through an HTTPS server). This image contains data processing software, all libraries it depends on, and the host-side IOValve libraries. Second, it resets the host (e.g., through the BMC). Lastly, it makes the host boot into the user-provided host image (e.g., through PXE network boot or boot storage emulation).

Attestation and secret provisioning. IOValve provisions secret keys to individual DPUs once it has remotely attested their operating systems. For example, the BlueField DPU supports device attestation based on the Security Protocols and Data Models (SPDM) standard [83]. IOValve can leverage it to enable remote attestation. IOValve uses the secret keys for two purposes. First, it assigns unique secret keys to individual DPUs such that the user as well as each DPU can distinguish the other. Second, it uses these keys to establish secure channels between DPUs to enable inter-node data processing and between the DPU and the user's on-premises machine to exchange confidential input data and processing results.

Security policy. IOValve enforces simple and strict security policies. First, IOValve blocks any unauthenticated or unencrypted traffic at the DPU. That is, all ingress and egress traffic to and from the node must be mutually authenticated and encrypted based on the provisioned secret keys. In essence, IOValve blocks all network traffic except for communications with the user's on-premises backend and a set of partner nodes explicitly specified (by their provisioned keys). Second, IOValve regularizes all inter-node (i.e., inter-DPU) traffic (which will be explained in §4.3 and §4.4). Third, if the user's on-premises machine does not support individual IOValve operations (e.g., due to a lack of DPU hardware), the DPU stages all confidential input and output data in its storage first and then delivers them to the node and the on-premises machines later, respectively, to avoid any potential information leakage.

4.3 Congestion-Free Memory Transfer

We explain how IOValve realizes memory transfer without timing channels due to host interconnect congestion [2]. In a computing system, many memory devices (e.g., DRAM and CPU cache) can become the basis for covert timing channels because multiple processors and peripherals can concurrently access them without performance isolation. For example, assume that a legitimate process is sending data from a communication buffer in DRAM to a remote party through a NIC which will access DRAM via DMA. Since a malicious process can also access DRAM, it can modulate information onto the legitimate data transfer by affecting the NIC's DMA transfer rate through memory congestion (§6.2). IOValve's congestion-free memory transfer satisfies the following properties, effectively mitigating the covert timing channel.

Spatial separation. IOValve spatially separates the communication buffer from the host CPU running untrusted code by placing it in the DPU (Fig. 2). The untrusted process cannot physically access the communication buffer, so it cannot intentionally introduce a memory access delay. To ensure this spatial separation, IOValve leverages the DPU memory with the following three hardware

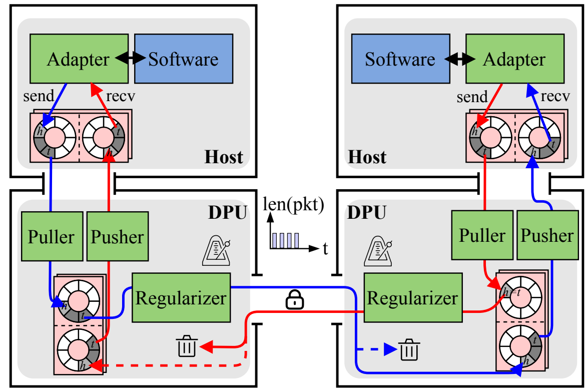


Fig. 2: IOValve's data transfer with congestion-free memory transfer and traffic regularization.

configuration steps. First, it prevents the host CPU (i.e., its DMA engine) from accessing DPU memory. Second, it prevents the NIC from accessing host memory. Third, it prevents the host CPU from accessing or congesting the link between the NIC and the DPU memory. Since the host CPU cannot access DPU memory whereas the NIC can only access DPU memory, IOValve requires a mechanism to let them share data in a leakage-free manner.

Periodic data pull and push. IOValve ensures periodic data pull and push between the host and DPU to prevent the host from arbitrarily congesting data transfers. IOValve runs the *puller* and *pusher* at the DPU which control the DPU's DMA engine. First, the puller and pusher periodically poll the metadata (i.e., tail and head indexes) of send and receive ring buffers in the host through DMA. If the host send buffer has a new data item and the DPU has a memory buffer to store it, the puller transfers the data item from the host send buffer to the DPU buffer (i.e., the puller is a consumer of the host send buffer). On the other hand, if the DPU has received a new data item from a peer DPU and the host receive buffer is available, the pusher transfers the data item from the DPU buffer to the host receive buffer (i.e., the pusher is a producer of the host receive buffer). IOValve separates the host send and receive buffers (i.e., they are unidirectional) and uses different DMA Submission Queues (SQs) to access them. This allows IOValve to concurrently run the puller and pusher without synchronization.

IOValve has the *adapter* in the host which allocates and manages the host send and receive ring buffers such that the puller and pusher can access and update them while interacting with untrusted software (as a compatibility layer §5). However, IOValve does not trust the adapter and prevents it from sending any DMA commands to the DPU. It is worth noting that the host might affect these periodic data pulls and pushes because the host memory that they are accessing can be under congestion (by a compromised adapter or any other host programs). Fortunately, IOValve's network traffic is independent of this congestion as its NIC does not access the host memory (§4.4).

Ring buffer lazy synchronization. IOValve shares host ring buffers across the host and DPU and it is challenging to ensure their consistency due to unpredictable memory ordering and nontrivial

Algorithm 1 Enqueue data with lazy synchronization

$m \leftarrow \text{metadata}(B, p)$
$m' \leftarrow \text{duplicate}(\text{metadata}(B, c))$
if $m'.s_b = m'.s_e$ **then**
  $m.t \leftarrow m'.t$ ▷ *synchronize metadata*
if $\text{available}(B)$ **then**
  $B[m.h] \leftarrow \text{data}$
  $m.s_b \leftarrow m.s_b + 1$
  $m.h \leftarrow m.h + 1 \bmod \text{capacity}(B)$
  $m.s_e \leftarrow m.s_b$

Algorithm 2 Dequeue data with lazy synchronization

$m \leftarrow \text{metadata}(B, c)$
$m' \leftarrow \text{duplicate}(\text{metadata}(B, p))$
if $m'.s_b = m'.s_e$ **then**
  $m.h \leftarrow m'.h$ ▷ *synchronize metadata*
if $\neg\text{empty}(B)$ **then**
  $\text{data} \leftarrow B[m.t]$
  $m.s_b \leftarrow m.s_b + 1$
  $m.t \leftarrow m.t + 1 \bmod \text{capacity}(B)$
  $m.s_e \leftarrow m.s_b$

transfer delay. If this was sharing of a ring buffer between host processes, IOValve could easily ensure consistency by leveraging CPU primitives like atomic operation, memory barrier, and mutex. However, in IOValve, the host and DPU feature different CPU architectures with different memory models (x86 and Arm) and share ring buffers through PCIe and DMAs. Our DPU, NVIDIA BlueField, provides a synchronization primitive called DOCA Sync Event [90] but we decided not to use it because it is slow, and we prefer a generic solution rather than a DPU-specific one.

IOValve solves this synchronization problem by allowing the host and DPU to lazily synchronize their metadata with a pair of sequence counters, inspired by sequential lock [116] and FaRM [33]. First, for each shared ring buffer (B), IOValve respectively maintains metadata in the host and DPU as a producer (p) or a consumer (c). Second, IOValve places sequence counters (s_b and s_e) at the lowest and highest addresses within the metadata ($s_b \leftarrow 0$ and $s_e \leftarrow 0$, initially). Third, IOValve ensures the host and DPU increase s_b before they update their metadata and do $s_e \leftarrow s_b$ after they have updated their metadata (e.g., with a memory barrier). For instance, once the adapter enqueues a new data item into the host send buffer based on the head index (h) in its metadata, it advances the head index (with a modular operation) while updating the sequence counters as explained. Similarly, once the puller dequeues a data item from the host send buffer based on its tail index (t), it advances the tail index while securing its metadata with the sequence counters. Fourth, IOValve ensures the DPU *sequentially* copies (from low to high memory addresses) the entire metadata of the host (and vice versa) before it attempts to synchronize the metadata (i.e., reflect a tail or head index advanced by the other party). The DPU discards copied metadata if $s_b \neq s_e$ as this implies that there has been a data race. Algorithm 1 and Algorithm 2 formalize these procedures. Since IOValve does not allow the host to access the DPU memory, it allocates an extra buffer in the host and pushes the DPU's metadata to the buffer to let the host do lazy synchronization.

It is worth noting that IOValve uses a pair of sequence counters rather than a single counter (and checks whether it is odd or even [116]) to decrease the number of DMA commands which are costly. Additionally, the untrusted host (i.e., a compromised adapter) might ignore this lazy metadata synchronization to intentionally introduce data inconsistency. However, this only results in integrity or availability issues which are out of this paper's scope.

Inter-buffer direct data transfer. IOValve directly copies data items from host send and receive buffers ($B_{h,s}$ and $B_{h,r}$) to DPU send and receive buffers ($B_{d,s}$ and $B_{d,r}$) and vice versa to avoid any redundant data copies and *bypass* the DPU CPU. More specifically, its DPU-side enqueue and dequeue operations for the ring buffer do not temporarily copy data items from and to another buffer with the CPU's memory copy instructions. Instead, IOValve calculates source and destination memory addresses (one belongs to the host memory whereas the other one belongs to the DPU memory) of a data item based on head and tail indexes and issues a DMA command to trigger direct data transfer between them. Algorithm 3 and Algorithm 4 formalize these procedures (along with a dummy remote transfer which will be explained in §4.4). Note that the DPU send buffer ($B_{d,s}$)—that the puller stores the host's data items to send to a peer DPU—is not shared with the host and the peer DPU. Thus, it does not require sequence counters and only needs CPU-level memory ordering unlike host send and receive buffers and DPU receive buffers.

Algorithm 3 Inter-buffer data pull (local)

$m \leftarrow \text{metadata}(B_{h,s}, c)$
$m' \leftarrow \text{duplicate}(\text{metadata}(B_{h,s}, p))$
if $m'.s_b = m'.s_e$ **then**
  $m.h \leftarrow m'.h$ ▷ *synchronize metadata*
if $\neg\text{empty}(B_{h,s}) \land \text{available}(B_{d,s})$ **then**
  $n \leftarrow \text{metadata}(B_{d,s}, p)$
  $B_{d,s}[n.h] \leftarrow B_{h,s}[m.t]$
  $m.s_b \leftarrow m.s_b + 1$
  $m.t \leftarrow m.t + 1 \bmod \text{capacity}(B_{h,s})$
  $m.s_e \leftarrow m.s_b$
  $n.h \leftarrow n.h + 1 \bmod \text{capacity}(B_{d,s})$

Node-level buffer management. IOValve maintains two buffers for each direction of a node pair regardless of the number of connections that the host software has established. While multiple connections between nodes can maximize data processing parallelism, this would allow the untrusted host code to affect IOValve's data transfer interval (e.g., by dynamically adjusting the number of connections during execution). It also increases the attack surface of the transferer due to its complexity. To this end, IOValve establishes only a single connection for each node pair and uses the two single-producer single-consumer ring buffers (i.e., one for each direction) for the node-level connection. The adapter at the host provides all other multiple connection support including maintaining multiple buffers for individual connections and transferring their data items into the node-level buffers in a synchronous manner.

Algorithm 4 Inter-buffer data push (remote or local)

$m \leftarrow \text{metadata}(B_{*,r}, p)$ ▷ $B_{*,r} = B_{d,r}$ (remote) or $B_{h,r}$ (local)
$m' \leftarrow \text{duplicate}(\text{metadata}(B_{*,r}, c))$
if $m'.s_b = m'.s_e$ **then**
  $m.t \leftarrow m'.t$ ▷ *synchronize metadata*
if $\neg\text{empty}(B_{d,*}) \wedge \text{available}(B_{*,r})$ **then** ▷ $B_{d,*} = B_{d,s}$ (remote) or $B_{d,r}$ (local)
  $n \leftarrow \text{metadata}(B_{d,*}, c)$
  $B_{*,r}[m.h] \leftarrow B_{d,*}[n.t]$
  $m.s_b \leftarrow m.s_b + 1$
  $m.h \leftarrow m.h + 1 \bmod \text{capacity}(B_{*,r})$
  $m.s_e \leftarrow m.s_b$
  if $B_{d,*} = B_{d,s} \wedge B_{*,r} = B_{d,r}$ **then**
    $n.t \leftarrow n.t + 1 \bmod \text{capacity}(B_{d,*})$
  else
    $n.s_b \leftarrow n.s_b + 1$
    $n.t \leftarrow n.t + 1 \bmod \text{capacity}(B_{d,*})$
    $n.s_e \leftarrow n.s_b$
else
  if $B_{d,*} = B_{d,s} \wedge B_{*,r} = B_{d,r}$ **then**
    $B_{*,r}[-1] \leftarrow B_{d,*}[-1]$ ▷ *dummy transfer*

4.4 Traffic Regularization

We explain how IOValve transfers data items remotely between DPUs without covert channels. The design of this secure remote data transfer largely overlaps with the congestion-free memory transfer between the host and the DPU (§4.3)—it periodically and directly transfers data items between ring buffers and synchronizes these buffers in a lazy manner. However, there are three critical differences in their implementation or security landscape. First, this remote data transfer uses RDMA instead of DMA. Second, this remote data transfer is explicitly observable by a covert channel listener (unlike the data transfer between the host and DPU). Third, this remote data transfer is between trusted entities (i.e., DPUs) which are cooperative.

Periodic remote data push. IOValve runs the *regularizer* at the DPU which enforces periodic remote data push to a peer DPU using one-sided RDMA Write and Read operations. Unlike two-sided RDMA Send and Recv operations, one-sided RDMA operations allow the sender to write and read data to and from the receiver's memory without involving the receiver's CPU (they must agree on which memory addresses are directly accessible during connection establishment). That is, the receiver's RNIC directly writes and reads (DMA) data to and from the receiver's memory addresses specified by the sender. In particular, the sender uses RDMA Read to poll the metadata of the peer's receive buffer for lazy synchronization. Also, it uses RDMA Write for remote inter-buffer direct data transfer and sharing its metadata of the peer's receive buffer with the peer for lazy synchronization. Rather than issuing two respective RDMA Write commands for a data item and metadata, IOValve uses RDMA Write with immediate to piggyback the metadata into the data transfer, reducing the number of RDMA operations.

Uniform data transfer. IOValve ensures uniform data transfer by sending out the same amount of data to a remote party at a fixed time interval. The sent data is either a real data item to be enqueued in the peer DPU's receive buffer or a fake data item to be discarded.

Table 1: Source Lines of Code (SLoC) of IOValve in C.

Component	SLoC	Component	SLoC
IOValve			
Adapter	258	Puller and pusher	805
Regularizer	893	Ring buffer	255
Compatibility layer			
NCCL Net plugin	350		

More specifically, the regularizer periodically checks its send buffer (filled by the puller) and the peer DPU's receive buffer (consumed by the pusher) to transfer a data item while padding the item with dummy data if it is smaller than the transfer unit. If the send buffer is empty or the peer's receive buffer is full, the regularizer transfers a dummy data item to the peer's dummy buffer which will be discarded (Algorithm 4). Due to remote reading of metadata, the data transfer pattern between DPUs is composed of small RDMA Read and large RDMA Write—both are uniformly sized. Since all network traffic between DPUs is protected with hardware-accelerated encryption, the network adversary cannot differentiate real data transfer from fake data transfer. Using uniform-size network packets and adjusting packet transmission intervals are well-known and effective countermeasures against traffic analysis [35, 127], IOValve adopts them in the context of a programmable network device.

Hardware-accelerated encryption. IOValve encrypts (uniform-size) messages before sending them onto the untrusted network to ensure confidentiality. IOValve must encrypt packets at the DPU because it does not trust the host. Since IOValve assumes low-latency and high-bandwidth networks (e.g., several 10s or 100s of Gbit/s), hardware acceleration is necessary to encrypt or decrypt them without performance degradation. If IOValve is configured to use InfiniBand, it should use InfiniBand-specific encryption mechanisms [101, 112, 128]. However, none of them is widely available. To this end, IOValve uses RDMA over Converged Ethernet version 2 (RoCEv2) [52] which transmits RDMA packets over UDP. We use the IPSec hardware acceleration to encrypt these UDP packets [85].

Persistent node-level connection. Another technique IOValve has adopted to regularize the network traffic is to persist node-level connections during data processing. That is, IOValve establishes per-node connections during system provisioning and maintains them until the system terminates (e.g., the user no longer uses it or the DPU gets reset). That is, IOValve does not establish connections based on the application's demand because it can result in covert channels. This node-level connection also avoids the communication overhead to re-establish connections [61].

5 Implementation

In this section, we explain how we implement IOValve. We do not cover components related to provisioning and attestation because we rely on existing software and libraries (e.g., running a web server at the DPU) to realize them. Table 1 shows the Source Lines of Code (SLoC) in the main components of our implementation of IOValve. The counts do not include any other library source files.

5.1 DPU Configuration

We configure the host and the IOValve DPU to satisfy the hardware requirements mentioned in §4.1. First, we do not install any extra NIC on our machine and do not connect Ethernet cables to the machine's motherboard-integrated NICs. Second, we directly plug our DPU into the machine's motherboard. That is, there are no additional PCIe switch devices. Third, we program the BlueField DPU's firmware (using `mlxconfig`) to make it use Zero-Trust mode [87] which ignores all administration commands from the host. Also, we enable the DPU's storage emulation and hide its RNIC from the host—the host can only see the BlueField DPU's DMA device. Fourth, we connect the BlueField DPU's out-of-band port to the machine's BMC port to let the DPU control the BMC. In addition, we confirm that no extra configuration is required to prevent the host from accessing the DPU memory because this is disallowed by default [1].

5.2 Congestion-Free Memory Transfer

We implement the three components of congestion-free memory transfer: the adapter, puller, and pusher. The adapter is a library for host applications (and compatibility layers) whereas the puller and pusher are threads running in the DPU. We leverage DOCA to implement them.

Adapter. The adapter provides two functions. The first allocates and maintains host-side memory buffers for DMA. It allocates page-aligned buffers for each node pair and registers them as DMA-able buffers using NVIDIA DOCA APIs (version: 2.2.0080) like `doca_mmap_set_memrange` and `doca_mmap_start`. The second creates ring buffers inside the DMA-able buffers and provides APIs to enqueue and dequeue data items or pointers to and from them. All data items and metadata of the ring buffers are cache line aligned.

Puller and pusher. The puller and pusher provide two functions. First, it allocates and maintains page-aligned DPU memory buffers for DMA and RDMA. It allocates enough buffer space for $2n$ unidirectional ring buffers where n is the number of communication partners specified by the user. Second, it provides APIs to copy data items from the host buffer to the DPU buffer and vice versa through DMA. More specifically, it uses DOCA's DMA APIs to enqueue DMA jobs to the DMA SQ (i.e., `doca_workq_submit`) and poll the DMA Completion Queue (CQ) (i.e., `doca_workq_progress_retrieve`).

5.3 Traffic Regularization

We implement the traffic regularizer as a DPU application consisting of multiple threads for inter-node connection management and memory transfer.

Connection management. The regularizer runs three threads for RDMA connection management: RDMA client, RDMA server, and RDMA CQ poller. The RDMA client and server threads establish two unidirectional RDMA Queue Pairs (QPs) for each node pair. Once all node-level channels have been established, they do nothing but maintain the established channels. The poller keeps polling the RDMA CQ to handle RDMA Send/Recv operations for connection establishment (e.g., share RDMA memory regions and corresponding RDMA keys). It also handles the completion of RDMA Write and Read operations to ensure the CQ never overflows. We use the `libibverbs` and `librdmacm` libraries (version: 2307mlnx47-1.2307050) to implement all RDMA-related functionality.

Remote memory transfer. The regularizer runs an RDMA sender thread for periodic remote data pushing. The sender performs two tasks for each peer node (round robin). First, it periodically checks the metadata of its send buffer and RDMA reads the metadata of a peer's receive buffer to check whether there is a new data item and to see whether the receiver can store it using `ibv_post_send` with `IBV_WR_RDMA_READ`. Second, it periodically RDMA writes data and metadata to the peer's memory using `ibv_post_send` with `IBV_WR_RDMA_WRITE_WITH_IMM`. If there is a new data item and the receive buffer is available, an RDMA operation uses valid source and destination addresses belonging to the send and receive ring buffers. Otherwise, it writes dummy data to a peer's memory region which is RDMA-able but will never be accessed by the peer.

5.4 NCCL Compatibility Layer

We implement a compatibility layer for NCCL [89] to run PyTorch [117] with IOValve. NCCL already specifies the NCCL Net plugin API [86] to let developers or users run NCCL on custom network environments. We write a NCCL plugin for IOValve which requires implementing essential functions such as `accept`, `connect`, `isend`, `irecv`, `listen`, and `test`. Specifically, our `isend` and `irecv` generate send and receive requests, and `test` attempts to enqueue or dequeue requests to or from corresponding host ring buffers until it succeeds.

6 Evaluation

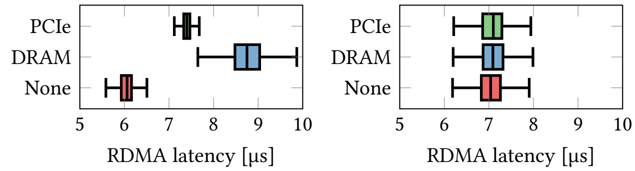
We evaluate IOValve by answering the following questions:

- **RQ1.** Is IOValve robust against covert channels? (§6.2)
- **RQ2.** How much network performance overhead does IOValve introduce? (§6.3)
- **RQ3.** What would be the realistic performance implication of IOValve when we use it for real-world applications like LLM fine-tuning, LLM batch inference, and molecular dynamics simulation (§6.4)?

6.1 Experimental Setup

Our experimental setup consists of two interconnected computing nodes. Each node features two Intel Xeon Silver 4510 processors (24 physical cores running at 2.4 GHz), 256 GiB of RAM, and 1 TB of NVMe SSD. It runs Ubuntu 22.04.5 LTS and its kernel version is 5.15. Each node is equipped with one NVIDIA BlueField-3 DPU with two up to 200 Gbit/s Ethernet/InfiniBand ports (B3220). Due to hardware constraints, we connect them directly through a single 100 Gbit/s Ethernet cable while enabling RoCEv2 [52] for RDMA over UDP (i.e., no InfiniBand). Our BlueField-3 DPU features 16 Arm cores running at 2.1 GHz, 16 GiB of RAM, and 128 GB of NVMe SSD. It runs Ubuntu 22.04.3 LTS whose kernel version is 5.15 (customized for BlueField). Both the BlueField DPU and the host machine use DOCA [90] for communication and data exchange. Also, to evaluate our system with real-world applications using GPUs, we install an

[1]https://forums.developer.nvidia.com/t/how-to-deal-with-dma-on-the-host-to-access-dpus-memory/221441



(a) RDMA with host memory **(b) RDMA with DPU memory**

Fig. 3: Comparison of RDMA Write latency with PCIe congestion, memory congestion, and no congestion.

NVIDIA TESLA P40 GPU with 24 GiB of memory on each node with driver version 550.127.05 and CUDA version 12.4. There is no peer-to-peer communication between the GPU and DPU.

6.2 Covert Channel Robustness

Because of single tenancy and traffic filtering at the DPU (§4.2), modulating the legitimate network traffic is the *only* covert channel that untrusted host code could potentially use to transmit data to the outside (§4.3). We evaluate whether IOValve is robust against network covert timing channels due to congestion at the only two hardware resources shared between the host and DPU: host memory and DPU's PCIe switch. Note that, as mentioned in §4.3 and §5.1, DPU memory congestion by the adversary is not possible such that we do not evaluate this aspect. To this end, we measure the RDMA Write latency of baseline (i.e., RDMA with host memory) and IOValve (i.e., RDMA with DPU memory) with and without intentional congestion. We focus on *pure* RDMA latency (i.e., without host-DPU DMA, traffic regularization, and CPU contention) because additional overhead would hide the congestion effect.

In all experiments, we run qperf [39] `rc_rdma_write_lat` with a 64 B payload (a cache line) between a pair of hosts and a pair of DPUs, respectively. One side (host or DPU) acts as the qperf client while the other side (DPU or host) acts as the qperf server. In the qperf RDMA Write latency test, the server prepares a buffer for RDMA and sleeps (i.e., no CPU involvement), and the client sends a packet to the server whose RNIC hardware immediately generates acknowledgments. The client measures the round-trip time upon receiving acknowledgments by polling the RDMA CQ. In our experiments, we measure the sensitivity of this round-trip time to various actions of the server host. We repeat each qperf measurement 1,000 times.

Host memory congestion. To show whether and how memory congestion affects the network traffic of the baseline and IOValve, we evaluate how a memory bandwidth benchmarking program [67], which fully stresses the DRAM at the host, affects RDMA latency. We let the program use all physical cores of the qperf server while measuring RDMA latency at the client. Fig. 3 shows the results. The host RDMA (baseline) latencies without and with memory congestion are 6.06 µs and 8.75 µs on average, respectively (Fig. 3a). That is, the RDMA latency with host memory increases by 44% on average when the host memory is congested, resulting in clearly observable inter-packet delay differences. In contrast, the host memory congestion does not affect the RDMA latency of IOValve (7.04 µs versus 7.09 µs Fig. 3b). Here, the RDMA latency with DPU memory is higher than that with host memory (regardless of congestion) because the DPU memory is slower than the host memory. The two-sample Kolmogorov-Smirnov statistic [16], which is a well-known measure to recognize covert channels [129], between the DPU RDMA latencies without and with host memory congestion is 0.069. In contrast, the same statistic for the host RDMA latency without and with host memory congestion is 1. Consequently, IOValve is robust against covert channels based on host memory congestion.

PCIe congestion. IOValve is also robust against covert channels due to PCIe congestion. Since IOValve prevents the host from directly accessing the RNIC and DPU memory, the host cannot easily congest the DPU's PCIe switch unlike existing PCIe congestion side channels [111] (i.e., no high-volume DMA exists). Instead, we write a program which repeatedly sends PCIe packets to the DPU by retrieving its PCIe configuration space. We run this program on the qperf server on all its physical cores while measuring the RDMA latency at the qperf client. The PCIe congestion program increases the host RDMA latency by 20% on average (Fig. 3a). In contrast, this program only increases the DPU RDMA latency by 0.85% which is within error bounds (Fig. 3b). The two-sample Kolmogorov-Smirnov statistic between the DPU RDMA latency without and with PCIe congestion is 0.09. In contrast, that between the host RDMA latency without and with PCIe congestion is 0.997. It is worth noting that even if an adversary figures out some other effective way to congest the DPU's PCIe switch from the host, we expect that these would not effectively modulate IOValve's network traffic. This is because the BlueField-3 DPU, that IOValve is based on, interconnects the RNIC, CPU, and PCIe switch via a coherent mesh (§2.2). That is, the RNIC and the CPU are directly connected such that their communication is negligibly affected by the PCIe switch.

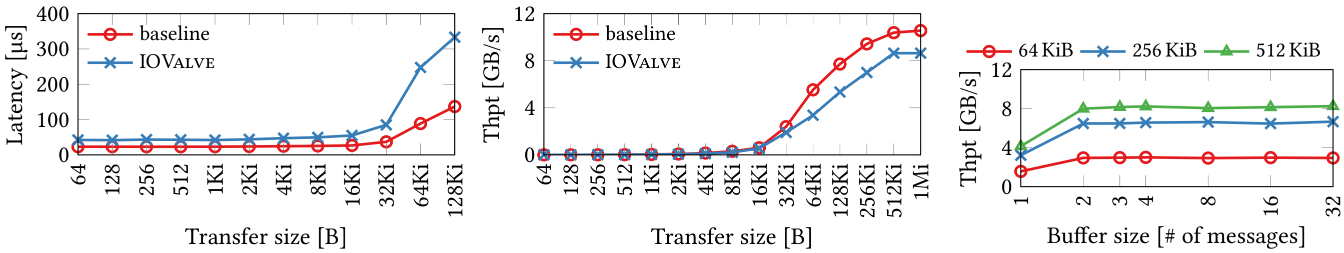
6.3 Micro-benchmark: Network Performance

We evaluate the latency and throughput of IOValve's message transfer as well as its collective communication performance. It is worth noting that this section only considers the network performance overhead of IOValve. Real-world workloads consist of both intra-node computation and inter-node communication such that the network performance overhead can be hidden especially if workloads are computationally expensive (§6.4).

We use the PyTorch communication benchmark [50] to measure the throughput of three collective operations: AllGather, AllReduce, and ReduceScatter. We additionally write two scripts based on this benchmark to measure round-trip latency and throughput. We run them with our two nodes using torchrun [98] which feature NCCL version 2.18.1 and PyTorch 2.1.1+cu121.

Latency and throughput. We compare the network latency (i.e., half the round-trip time) and throughput of IOValve against the stock NCCL with host RDMA. We vary the unit transfer size from 64 B to 128 KiB for the latency measurement (with the `isend` function) and from 64 B to 1 MiB for the throughput measurement (with the `broadcast` function) [2]. We repeat each measurement 100 times and average them. Fig. 4 shows the results. The latency of IOValve is 77.4%–2.8× higher than that of the stock NCCL and its throughput is 16.8%–39.1% lower than that of the stock NCCL. Neither stock NCCL nor IOValve benefits from a transfer unit greater than

[2] PyTorch's `isend` did not allow us to transfer a single tensor larger than 128 KiB so we were not able to test the latency with larger than 128 KiB transfer units.



(a) Latency **(b) Throughput** **(c) IOValve throughput by buffer size**

Fig. 4: Comparison of the network latency and throughput between IOValve and stock NCCL (baseline).

512 KiB. IOValve's high latency is expected because it copies a data item from a host to a DPU, between DPUs, and then from a DPU to a host in stages (two DMA and one RDMA operations) whereas the stock NCCL directly copies a data item between hosts using a single RDMA operation. If there are one or more hops (e.g., RDMA switches) between DPUs, IOValve's latency overhead would decrease. In contrast, IOValve's throughput overhead is reasonably low even though there are dummy transfers and host-to-DPU and DPU-to-DPU communications are loosely synchronized.

We evaluate how the ring buffer size affects network performance for various message or unit transfer sizes (i.e., 64, 256, and 512 KiB). The throughput of IOValve is largely independent of the buffer size (Fig. 4c). This is because IOValve maintains a separate ring buffer for each unidirectional connection, and a node waits for an acknowledgment for its old real message from the peer before sending a new real message. This acknowledgment traffic is also regularized. IOValve dequeues and transfers one real (if it exists) or dummy message at a time (i.e., there is no batching), so its latency is independent of the buffer size.

Collective communication. We compare IOValve's collective communication performance against the stock NCCL with host RDMA. We configure IOValve to use two transfer sizes: 256 KiB and 64 KiB. We set the `NCCL_BUFFSIZE` environment variable accordingly to let NCCL break down messages larger than 256 KiB or 64 KiB. We do not adjust the transfer size of the stock NCCL. Fig. 5 shows the results. As expected, IOValve's throughput overhead is high if the data size is small (e.g., smaller than 2 MB). Its throughput is 60.9%–6.1× (with 256 KiB transfer) and 30%–3.5× (with 64 KiB transfer) lower than that of the stock NCCL. This is expected because IOValve does not use variable-length transfer (unlike the stock NCCL) and its transfer units are mostly empty in those cases. However, when the data size is between 2 MB and 30 MB, IOValve's throughput is 34.1%–90.4% lower than that of the stock NCCL. Also, when the data size is larger than 30 MB, IOValve's throughput is 18.7%–39.1% lower than that of the stock NCCL. These numbers are based on 256 KiB transfers. One way to avoid this problem is to use various transfer unit sizes to minimize the traffic wastage but this weakens the security of IOValve—a further study is necessary to figure out tolerable information leakage.

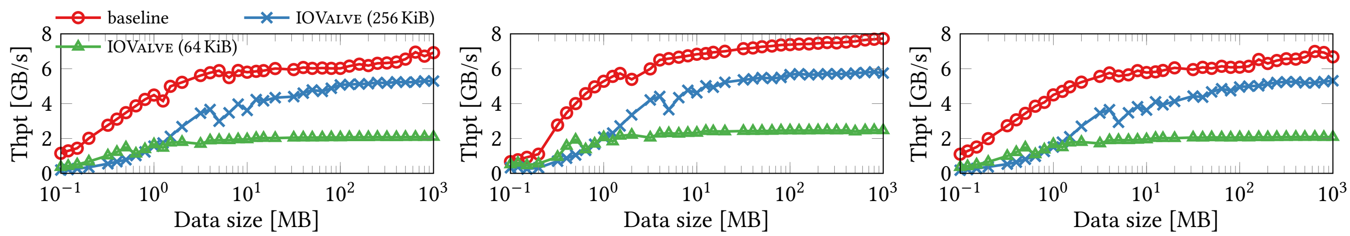
In addition, we evaluate the NCCL throughput of IOValve with transfer sizes of 512 KiB and 1 MiB and confirm that they are worse than that with 256 KiB transfer. This is because their marginal throughput improvement does not compensate for their high latency overhead. To this end, we decide to use 256 KiB as the default transfer size of the remaining experiments (§6.4).

6.4 Real-World Applications

We evaluate IOValve on three real-world applications: LLM fine-tuning, LLM batch inference, and molecular dynamics.

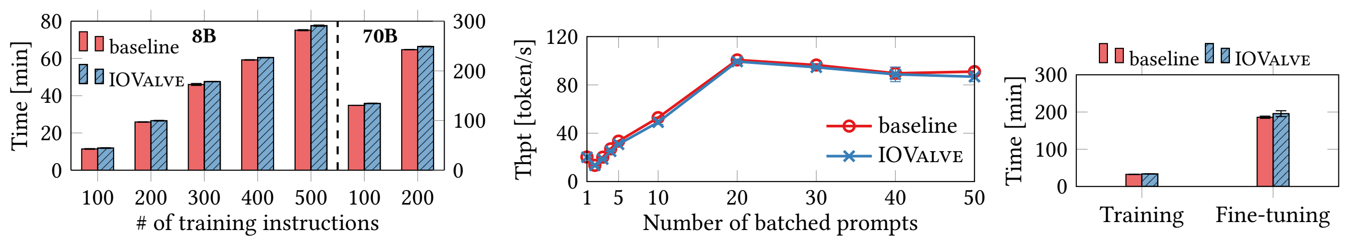
LLM fine-tuning. We evaluate the performance of IOValve for LLM fine-tuning which adjusts the parameters of a pre-trained general-purpose LLM with custom (and possibly confidential) data. To this end, we fine-tune Llama 3 [34] using an open-source fine-tuning script [105] which relies on Hugging Face Transformers [46], PyTorch Fully Sharded Data Parallel (FSDP) [139], and QLoRA [32] with the 4-bit NormalFloat (NF4) quantization. We configure the script to use the half-precision floating point format (FP16). We fine-tune the Llama 3 8B model with 100–500 training instructions and the Llama 3 70B model with 100–200 training instructions randomly sampled from the No Robots dataset [99] where each training instruction consists of system, user, and assistant prompts. We compare the performance of IOValve over the baseline where we use the same script with the stock NCCL plugin and host RDMA. We repeat the experiments five to ten times while measuring total training time. We confirm that IOValve marginally increases fine-tuning training time by 2.1%–4.1% as shown in Fig. 6a. The performance overhead of IOValve decreases as we increase the model size, the number of training instructions, or both, which results in high computation overhead—fine-tuning spends more time on using CPUs and GPUs before transmitting computation results through a NIC.

LLM batch inference. We evaluate the performance of IOValve for batch inference that collects and processes several inputs (or prompts) together to improve token generation throughput [63], which is useful for both online and offline inference [93]. We write a distributed batch inference script using Hugging Face Transformers [46]. This script fits the Llama 3 8B model [34] on our two GPUs with tensor parallelism. We use neither sampling nor key-value cache for more deterministic results. We use prompts randomly sampled from the Argilla's ShareGPT dataset [10] and let our script generate up to 50 tokens for each prompt. We repeat the experiments ten times while measuring their token generation throughput. Fig. 6b shows the evaluation results. The token generation throughput of the batch inference with IOValve is 0.2%–8.5% lower than that with the stock NCCL when the number of batched prompts



(a) AllGather (b) AllReduce (c) ReduceScatter

Fig. 5: Comparison of collective communication throughput between IOVALVE and stock NCCL (baseline).



(a) Llama 3 fine-tuning (b) Batch inference's token generation (c) DeePMD training and fine-tuning

Fig. 6: Comparison of the performance of real-world applications between IOVALVE and stock NCCL (baseline).

lies between 1 and 50. Both achieve the highest throughput at the batch size of 20 where IOVALVE's throughput is 1.4% lower than that of the stock NCCL. State of the art LLM serving systems like vLLM [63] focus on achieving high throughput with a large batch size and IOVALVE aligns with this.

Molecular dynamics. We evaluate the performance of IOVALVE for DeePMD [124] which enables deep learning for molecular dynamics to simulate the movement of atoms and molecules over time. DeePMD supports PyTorch Distributed Data Parallel (DDP) [70] as its backend. We use its incorporated example and dataset (`water/se_e2_a`) to train a small deep learning model and to fine-tune a foundation model [137]. We repeat the experiments ten times. Fig. 6c shows the results. IOVALVE marginally increases training and fine-tuning times by 3.4% and 5.1%, respectively. Like LLM fine-tuning and batch inference, DeePMD is computationally expensive such that IOVALVE does not incur high overhead.

7 Discussion

In this section, we discuss design alternatives and limitations.

Alternative design: Middlebox. IOVALVE is a network interposer between a node and the outside. This makes a middlebox [11, 129] and a sidecar [36] potential candidates for realizing IOVALVE. A middlebox is a bump-in-the-wire machine which is typically a programmable switch [11, 129] or a general-purpose computer. We do not consider a middlebox-based IOVALVE because a middlebox typically manages multiple nodes (e.g., in the same rack as a Top-of-Rack (ToR) switch) and is operated by the cloud provider. Assuming a per-node middlebox with full user control is less practical. Also, the middlebox should process or filter the arbitrary packets generated by the host, potentially resulting in resource congestion which leads to covert channels.

Alternative design: Sidecar. A proxy or sidecar [36] is attached to a target instance (a container or VM) to manipulate its behavior (e.g., filtering and encrypting its network traffic). We do not consider a sidecar-based IOVALVE due to the following two reasons. First, the sidecar relies on the underlying operating system or hypervisor to interpose on the network traffic of the target node (or instance). However, IOVALVE does not trust any code running inside the node. Second, the sidecar is vulnerable to covert channels because it is running inside the node which also runs the untrusted target instance—i.e., there is no spatial separation. To this end, we focus on the DPU-based IOVALVE design in this paper.

Interface and protocol robustness. Hardening I/O interface and protocol for confidential or untrusted computing is a challenging problem [66]. IOVALVE overcomes this problem using a static interface and a simple protocol. That is, it does not support variable-length messages and various message types which can result in memory safety vulnerabilities. It copies fixed-size messages from a fixed set of known memory addresses (i.e., fixed-size ring buffer slots) of a host to a DPU and transfers them to a fixed set of peers which are determined during provisioning, thereby making its interface and protocol difficult to misimplement.

Limitation. One limitation of IOVALVE is that it is less suitable for dynamic and latency-sensitive data processing like chatbots. Technically, we can enforce traffic regularization between the user's on-premises machine and the DPU (like the one between DPUs) to securely exchange prompts and responses, mitigating the token-length side-channel attack [126]. However, since the lengths of prompts and responses are largely unpredictable, we should sufficiently enlarge the unit transfer size which suffers from non-negligible network overhead. We leave this to future work.

Deployability and compatibility. IOValve requires bare metal instances with DPUs and does not apply to VMs. However, we target large-scale computations spanning multiple nodes with GPUs for which bare metal instances are a better fit than VMs. Major cloud providers offer DPUs (§2.2) and provide bare-metal cloud services based on them [7]. IOValve imposes no restrictions on the host software stack beyond requiring NCCL support. As distributed GPU computations require NCCL along with CUDA, this should involve no extra effort. In addition, IOValve requires a network plugin on the host which is analogous to a user-mode NIC driver.

8 Related Work

Sandboxes. A long line of systems aims to protect sensitive data from the software that processes them [6, 15, 29, 31, 54, 58, 62, 72, 75, 92, 104, 108, 121, 125, 133, 134]. This work goes back to at least the 1960s and 1970s and spans many generations of hardware and software. A common theme is fine-grained hardware sharing between the untrusted software and the software TCB. Examples include untrusted processes and a trusted operating system [62, 108, 121, 133, 134] or untrusted VMs and a trusted hypervisor [15, 58, 104] sharing the same processor cores. As a result of this fine-grained sharing, these systems contend with an intractable multitude of hardware side and covert channels [140]. The VAX VMM Security Kernel included serious efforts to eliminate and mitigate hardware side channels but was only partially successful [58]. More recently, the flood of microarchitectural side channels [20, 73] raises fundamental questions about the possibility of avoiding covert channels under fine-grained hardware sharing.

Several systems limit hardware sharing by partitioning processor cores [21, 140] or other microarchitectural resources [122]. While this eliminates many of the known side channels, the remaining shared hardware can still be used to construct covert channels. IOValve avoids these problems by moving sandbox enforcement to what is effectively a separate computer.

Confidential untrusted computing. Starting with Ryoan [48], several systems move the sandbox into the cloud and use SGX enclaves to protect it from the cloud provider [3, 47, 48, 95]. These systems inherit key security and performance properties from SGX.

First, fine-grained sharing of processor hardware between an untrusted operating system and the sandbox inside an enclave provides an abundance of hardware side channels [17, 22, 42, 45, 65, 68, 69, 79, 97, 109, 120] that can be used to build covert channels. Unsurprisingly, mitigations in these systems focus on software-based covert channels while hardware covert channels are declared out of scope and ignored. In contrast, IOValve moves the sandbox boundary to where these channels do not even exist.

Second, the sandbox inside an SGX enclave is an application running on an untrusted operating system. This prevents direct hardware access by the sandbox and introduces transitions into and out of enclaves as a source of overhead. Lack of support for intra-enclave isolation by SGX forces these systems to use software techniques (e.g., SFI [123] and NaCl [131]), incurring additional overhead. None of the systems was evaluated with high-performance hardware. Chiron [47] reports CIFAR training times of several hours—a task that has been completed in seconds on an A100 GPU [57]. IOValve is not subject to any of these limitations, as it runs natively on the host with unencumbered hardware access.

Hecate [38] and Erebor [136] place the sandbox inside AMD SEV-SNP and Intel TDX CVMs which hold the promise of better hardware access. DLBox [49] and Continuum AI [74] let the sandbox access a GPU. These sandboxes could benefit from TDISP [96] and NVLink encryption [91] for secure and efficient device access once they are available. On the other hand, CVMs retain SGX's fine-grained sharing of processor hardware. This leaves them with the same intractable covert channel problem as in the SGX case.

Confidential large-scale computing. Several recent studies have considered how to realize confidential computing over multiple nodes. HETEE [141] leverages a recent technology called the PCIe ExpressFabric to program the PCIe connection between nodes and accelerators (e.g., GPUs) with an external controller. HETEE and IOValve can benefit from each other because HETEE has no security mechanism against covert channels and IOValve can leverage HETEE's programmable spatial separation. Akram et al. [4] and Mohan et al. [80] study whether existing confidential CPU and GPU technologies are suitable for securing HPC and AI workloads. They conclude that core-level execution, large TCB, slow CPU-GPU interaction due to encryption, and a lack of side-channel mitigation result in insecure or unusable systems. In contrast, IOValve does not suffer from the issues they mention.

Network side-channel mitigation. Traffic analysis can reveal or fingerprint traffic content regardless of whether it is encrypted [19, 35, 76, 77, 103, 127, 138]. Network side-channel mitigations cope with this threat by shaping a victim's benign traffic to make it robust against side-channel analysis. Instead of using constant or fixed shaping which is known to be secure but incurs overhead for bursty traffic [35], these schemes propose traffic-aware adaptive shaping mechanisms with statistical privacy guarantees [19] or using differential privacy [103]. However, they generally exclude covert channels from their threat model. IOValve uses constant-rate shaping instead of adaptive mechanisms because statistical privacy is difficult to ensure against malicious applications and covert-channel attackers [43, 77, 103]. Also, IOValve targets collective communication which has almost uniform traffic patterns (§2.3).

9 Conclusion

There is an ever growing demand for large-scale computation on confidential data. However, it is difficult to maintain data confidentiality while trusting neither the compute infrastructure nor the full software stack. The former can be astronomically expensive, and the latter is difficult to achieve due to complicated software supply chains. IOValve addresses this critical security problem by rethinking the perimeter of confidential computing: it runs untrusted software on bare-metal machines in the public cloud whose external interactions (i.e., network I/O) are strictly encrypted, filtered, and regularized by separate hardware components (i.e., DPUs) to defeat both overt- and covert-channel attacks. Our evaluation shows that IOValve has marginal performance overhead and supports real-world applications (i.e., LLM fine-tuning and batch inference, and molecular simulation).

Acknowledgments

We would like to thank Paul England for his insight and support. We also thank the anonymous reviewers for their helpful feedback.

References

[1] Advanced Micro Devices, Inc. 2024. AMD Pensando Networking. https://www.amd.com/en/products/accelerators/pensando.html.

[2] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher De Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. 2022. Understanding Host Interconnect Congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks (HotNets)*.

[3] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. 2021. Chancel: Efficient Multi-client Isolation Under Adversarial Programs. In *Proceedings of the 2021 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[4] Ayaz Akram, Venkatesh Akella, Sean Peisert, and Jason Lowe-Power. 2022. SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*.

[5] Rahaf Alkhadra, Joud Abuzaid, Mariam AlShammari, and Nazeeruddin Mohammad. 2021. Solar Winds Hack: In-Depth Analysis and Countermeasures. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*.

[6] Kalev Alpernas, Cormac Flanagan, Sadjad Fouladi, Leonid Ryzhyk, Mooly Sagiv, Thomas Schmitz, and Keith Winstein. 2018. Secure serverless computing using dynamic information flow control. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018).

[7] Amazon Web Services, Inc. 2024. The Security Design of the AWS Nitro System. https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.html.

[8] AMD. 2021. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[9] Anyiscale, Inc. 2024. Productionizing and scaling Python ML workloads simply – Ray. https://www.ray.io.

[10] Argilla. 2023. Dataset Card for sharegpt-text-descriptives. https://huggingface.co/datasets/argilla/sharegpt-text-descriptives.

[11] Manikandan Arumugam, Deepak Bansal, Navdeep Bhatia, James Boerner, Simon Capper, Changhoon Kim, Sarah McClure, Neeraj Motwani, Ranga Narasimhan, Urvish Panchal, Tommaso Pimpo, Ariff Premji, Pranjal Shrivastava, and Rishabh Tewari. 2022. Bluebird: High-performance SDN for Bare-metal Cloud Services. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Renton, WA.

[12] Yechan Bae, Sarbartha Banerjee, Sangho Lee, and Marcus Peinado. 2022. Spacelord: Private and Secure Smart Space Sharing. In *Proceedings of the 38th Annual Computer Security Applications Conference (ACSAC)*.

[13] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Broomfield, Colorado.

[14] David E. Bell and Leonard La Padula. 1976. *Secure Computer System: Unified Exposition and Multics Interpretation*. Technical Report MTR-2997, Rev. 1. MITRE Corporation.

[15] Stefan Berger, Ramón Cáceres, Dimitrios Pendarakis, Reiner Sailer, Enriquillo Valdez, Ronald Perez, Wayne Schildhauer, and Deepa Srinivasan. 2008. TVDc: Managing Security in the Trusted Virtual Datacenter. *ACM SIGOPS Operating Systems Review* 42, 1 (2008), 40–47.

[16] Vance W Berger and YanYan Zhou. 2014. Kolmogorov–Smirnov test: Overview. *Wiley statsref: Statistics reference online* (2014).

[17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *Proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT)*.

[18] Robert Buhren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. 2021. One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization. In *Proceedings of the 28th ACM Conference on Computer and Communications Security (CCS)*. Virtual.

[19] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. Scottsdale, Arizona.

[20] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *Proceedings of the 28th USENIX Security Symposium (Security)*. Santa Clara, CA.

[21] Charly Castes and Andrew Baumann. 2024. Sharing is leaking: blocking transient-execution attacks with core-gapped confidential VMs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. La Jolla, CA.

[22] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *Proceedings of the 4th IEEE European Symposium on Security and Privacy (Euro S&P)*.

[23] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D Garcia. 2021. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In *Proceedings of the 30th USENIX Security Symposium (Security)*. Virtual.

[24] Lucian Constantin. 2022. Supply Chain Attacks Increased Over 600% This Year and Companies Are Falling Behind. https://www.csoonline.com/article/573925/supply-chain-attacks-increased-over-600-this-year-and-companies-are-/falling-behind.html.

[25] Intel Corporation. 2021. Intel® Trust Domain Extensions White Paper. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf.

[26] Franck Courbon, Sergei Skorobogatov, and Christopher Woods. 2016. Reverse Engineering Flash EEPROM Memories Using Scanning Electron Microscopy. In *International Conference on Smart Card Research and Advanced Applications*. 57–72.

[27] Cybersecurity and Infrastructure Security Agency. 2025. Defining Insider Threats. https://www.cisa.gov/topics/physical-security/insider-threat-mitigation/defining-insider-threats.

[28] Datanyze. 2025. Google Cloud Company Profile. https://www.datanyze.com/companies/google-cloud/356413659.

[29] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. 2020. Valve: Securing Function Workflows on Serverless Computing Platforms. In *Proceedings of The Web Conference (WWW)*.

[30] John Kwao Dawson, Frimpong Twum, James Benjamin Hayfron Acquah, and Yaw Marfo Missah. 2023. Ensuring confidentiality and privacy of cloud data using a non-deterministic cryptographic scheme. *PLOS ONE* 18, 2 (2023), e0274628.

[31] Dorothy E. Denning and Peter J. Denning. 1977. Certification of Programs for Secure Information Flow. *Commun. ACM* 20, 7 (July 1977), 504–513.

[32] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[33] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Seattle, WA.

[34] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 herd of models. arXiv:2407.21783

[35] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[36] Envoy Project Authors. 2024. Envoy proxy. https://www.envoyproxy.io.

[37] Yue Gao, Ilia Shumailov, and Kassem Fawaz. 2025. Supply-Chain Attacks in Machine Learning Frameworks. In *Proceedings of the Eighth Annual Conference on Machine Learning and Systems (MLSys)*.

[38] Xinyang Ge, Hsuan-Chi Kuo, and Weidong Cui. 2022. Hecate: Lifting and Shifting On-Premises Workloads to an Untrusted Cloud. In *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*. Los Angeles, CA.

[39] Johann George. [n. d.]. qperf - Measure RDMA and IP performance. https://linux.die.net/man/1/qperf.

[40] Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, John C. Mitchell, and Alejandro Russo. 2012. Hails: Protecting Data Privacy in Untrusted Web Applications. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Hollywood, CA.

[41] Dan Goodin. 2024. What we know about the xz Utils backdoor that almost infected the world. *Ars Technica* (March 2024). https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/

[42] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on System Security (EuroSec)*.

[43] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. 2011. Differential Privacy Under Fire. In *Proceedings of the 20th USENIX Security Symposium (Security)*. San Francisco, CA.

[44] Steven Herschbein, Shida Tan, Richard Livengood, and Michael Wong. 2022. An Introduction to the FIB as a Microchip Circuit Edit Tool. In *International Symposium for Testing and Failure Analysis*.

[45] Felicitas Hetzelt and Robert Buhren. 2017. Security Analysis of Encrypted Virtual Machines. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*.

[46] Hugging Face. [n. d.]. Transformers. https://huggingface.co/docs/transformers/en/index.

[47] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. arXiv:1803.05961

[48] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, GA.

[49] Jaewon Hur, Juheon Yi, Cheolwoo Myung, Sangyun Kim, Youngki Lee, and Byoungyoung Lee. 2025. DLBox: New Model Training Framework for Protecting Training Data. In *Proceedings of the 2025 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[50] IBM. 2024. PyTorch communication benchmarks. https://github.com/IBM/pytorch-communication-benchmarks.

[51] Immuta. 2021. Survey Reveals Emerging Challenges with Data Security, Privacy Amid Shift to the Cloud. *TDWI* (2021). https://tdwi.org/articles/2021/11/02/immuta-survey-news.aspx

[52] Infiniband Trade Association. 2014. RoCEv2. https://cw.infinibandta.org/document/dl/7781.

[53] Intel Corporation. 2024. Intel Infrastructure Processing Unit (Intel IPU). https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.

[54] Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, and Michael Swift. 2023. Guarding Serverless Applications with Kalium. In *Proceedings of the 32nd USENIX Security Symposium (Security)*. Anaheim, CA.

[55] JJ. Kathuria and Arjun Bhardwaj. 2022. Insider Threat: Impact Studies. https://cloud.google.com/blog/products/identity-security/insider-threat-impact-studies/.

[56] JJ Kathuria and Arjun Bhardwaj. 2022. Insider Threat: The Dangers Within. https://cloud.google.com/blog/products/identity-security/insider-threat-dangers-within/.

[57] Keller Jordan. 2024. 94% on CIFAR-10 in 3.29 Seconds on a Single GPU. arXiv:2404.00498

[58] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. 1991. A Retrospective on the VAX VMM Security Kernel. *IEEE Transactions on Software Engineering* 17, 11 (1991), 1147–1165.

[59] Uri Katz, Guy Kaplan, and Avi Lumelsky. 2024. Shelltorch Explained: Multiple Vulnerabilities in PyTorch Model Server (Torchserve) (CVSS 9.9, CVSS 9.8) Walkthrough. https://www.oligo.security/blog/shelltorch-explained-multiple-vulnerabilities-in-pytorch-model-server.

[60] Shaharyar Khan, Ilya Kabanov, Yunke Hua, and Stuart Madnick. 2022. A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned. *ACM Transactions on Privacy and Security* 26, 1 (2022), 1–29.

[61] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. 2019. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, MA.

[62] Max Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information Flow Control for Standard OS Abstractions. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*. Stevenson, WA.

[63] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*. Koblenz, Germany.

[64] Kevin Lee, Adi Gangidi, and Mathew Oldham. 2024. Building Meta's GenAI Infrastructure. https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/.

[65] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *Proceedings of the 26th USENIX Security Symposium (Security)*. Vancouver, Canada.

[66] Hugo Lefeuvre, David Chisnall, Marios Kogias, and Pierre Olivier. 2023. Towards (Really) Safe and Fast Confidential I/O. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HotOS)*.

[67] Daniel Lemire. 2024. Estimating your memory bandwidth. https://lemire.me/blog/2024/01/13/estimating-your-memory-bandwidth/.

[68] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. 2022. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[69] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. 2019. Exploiting Unprotected I/O Operations in AMD's Secure Encrypted Virtualization. In *Proceedings of the 28th USENIX Security Symposium (Security)*. Santa Clara, CA.

[70] Shen Li. 2019. Getting Started with Distributed Data Parallel. https://docs.pytorch.org/tutorials/intermediate/ddp_tutorial.html.

[71] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2023. Enabling Realms with the Arm Confidential Compute Architecture. *;login: The USENIX Magazine* (July 2023).

[72] Peter Loscocco and Stephen Smalley. 2001. Integrating Flexible Support for Security Policies into the Linux Operating System. In *Proceedings of the 2001 USENIX Annual Technical Conference—FREENIX Track*.

[73] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. 2021. A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–37.

[74] Laura Martinez. 2024. Advancing Security for Large Language Models with NVIDIA GPUs and Edgeless Systems. https://developer.nvidia.com/blog/advancing-security-for-large-language-models-with-nvidia-gpus-and-edgeless-systems/.

[75] Jonathan M. McCune, Trent Jaeger, Stefan Berger, Ramón Cáceres, and Reiner Sailer. 2006. Shamon: A System for Distributed Mandatory Access Control. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[76] Aastha Mehta, Mohamed Alzayat, Roberta De Viti, Björn B Brandenburg, Peter Druschel, and Deepak Garg. 2022. Pacer: Comprehensive Network Side-Channel Mitigation in the Cloud. In *Proceedings of the 31st USENIX Security Symposium (Security)*. Boston, MA.

[77] Roland Meier, Vincent Lenders, and Laurent Vanbever. 2022. ditto: WAN Traffic Obfuscation at Line Rate. In *Proceedings of the 2022 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[78] Microsoft. 2024. Overview of Azure Boost. https://learn.microsoft.com/en-us/azure/azure-boost/overview.

[79] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In *Cryptographic Hardware and Embedded Systems (CHES)*.

[80] Apoorve Mohan, Mengmei Ye, Hubertus Franke, Mudhakar Srivatsa, Zhuoran Liu, and Nelson Mimura Gonzalez. 2024. Securing AI Inference in the Cloud: Is CPU-GPU Confidential Computing Ready?. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*.

[81] Amin Mosayyebzadeh, Apoorve Mohan, Sahil Tikale, Mania Abdi, Nabil Schear, Charles Munson, Trammell Hudson, Larry Rudolph, Gene Cooperman, Peter Desnoyers, and Orran Krieger. 2019. Supporting Security Sensitive Tenants in a Bare-Metal Cloud. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. Renton, WA.

[82] National Insider Threat Special Interest Group. 2024. 2024 Insider Threat Incidents Report for the Department of Defense. https://cloud.google.com/blog/products/identity-security/insider-threat-dangers-within/.

[83] NVIDIA Corporation. 2023. NVIDIA Device Attestation and CoRIM-based Reference Measurement Sharing. https://docs.nvidia.com/networking/display/ndacrmsv10/introduction.

[84] NVIDIA Corporation. 2024. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.

[85] NVIDIA Corporation. 2024. IPSec Full Offload. https://docs.nvidia.com/networking/display/mlnxofedv24010331/ipsec+full+offload.

[86] NVIDIA Corporation. 2024. NCCL Net Plugin Documentation. https://github.com/NVIDIA/nccl/blob/master/ext-net/README.md.

[87] NVIDIA Corporation. 2024. NVIDIA BlueField Modes of Operation. https://docs.nvidia.com/doca/sdk/nvidia+bluefield+modes+of+operation/index.html.

[88] NVIDIA Corporation. 2024. NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[89] NVIDIA Corporation. 2024. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.

[90] NVIDIA Corporation. 2024. NVIDIA DOCA Software Framework. https://developer.nvidia.com/networking/doca.

[91] NVIDIA Corporation. 2025. The NVIDIA Grace Blackwell Superchip. https://docs.nvidia.com/multi-node-nvlink-systems/multi-node-tuning-guide/.

[92] Department of Defense. 1985. *Trusted Computer System Evaluation Criteria (Orange Book)*. Technical Report DoD 5200.28-STD.

[93] OpenAI. 2024. Batch API - OpenAI API. https://platform.openai.com/docs/guides/batch.

[94] Optrium. 2025. How Many People Are Needed to Run a Data Centre? https://optrium.co.uk/how-many-people-are-needed-to-run-a-data-centre/.

[95] Minkyung Park, Jaeseung Choi, Hyeonmin Lee, and Taekyoung Kwon. 2025. PAVE: Information Flow Control for Privacy-preserving Online Data Processing Services. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Rotterdam, The Netherlands.

[96] PCI-SIG. 2022. TEE Device Interface Security Protocol (TDISP). https://pcisig.com/tee-device-interface-security-protocol-tdisp.

[97] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Capkun. 2021. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In *Proceedings of the 30th USENIX Security Symposium (Security)*. Virtual.

[98] PyTorch Contributors. 2023. torchrun (Elastic Launch). https://pytorch.org/docs/stable/elastic/run.html.

[99] Nazneen Rajani, Lewis Tunstall, Edward Beeching, Nathan Lambert, Alexander M. Rush, and Thomas Wolf. 2023. No Robots. https://huggingface.co/datasets/HuggingFaceH4/no_robots.

[100] Rhonda Ascierto and Todd Traver. 2021. *Data center security: Reassessing physical, human and digital risks.* Technical Report.

[101] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. 2021. ReDMArk: Bypassing RDMA Security Mechanisms. In *Proceedings of the 30th USENIX Security Symposium (Security)*. Virtual.

[102] Indrajit Roy, Srinath T.V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. 2010. Airavat: Security and Privacy for MapReduce. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. San Jose, CA.

[103] Amir Sabzi, Rut Vora, Swati Goswami, Margo Seltzer, Mathias Lécuyer, and Aastha Mehta. 2024. NetShaper: A Differentially Private Network Side-Channel Mitigation System. In *Proceedings of the 33rd USENIX Security Symposium (Security)*. Philadelphia, PA.

[104] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramón Cáceres, Ronald Perez, Stefan Berger, John Linwood Griffin, and Leendert van Doorn. 2005. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[105] Philipp Schmid. 2024. Efficiently fine-tune Llama 3 with PyTorch FSDP and Q-Lora. https://www.philschmid.de/fsdp-qlora-llama3.

[106] Philipp Schmid, Omar Sanseviero, Alvaro Bartolome, Leandro von Werra, Daniel Vila, Vaibhav Srivastav, Marc Sun, and Pedro Cuenca. 2024. Llama 3.1 - 405B, 70B & 8B with multilinguality and long context. https://huggingface.co/blog/llama31.

[107] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud using SGX. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*. San Jose, CA.

[108] Helgi Sigurbjarnarson, Luke Nelson, Bruno Castro-Karney, James Bornholt, Emina Torlak, and Xi Wang. 2018. Nickel: A Framework for Design and Verification of Information Flow Control Systems. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Carlsbad, CA.

[109] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. 2019. MicroScope: Enabling Microarchitectural Replay Attacks. In *Proceedings of the 46th Annual International Symposium on Computer Architecture (ISCA)*.

[110] John Stawinski. 2024. Playing with Fire - How We Executed a Critical Supply Chain Attack on PyTorch. https://johnstawinski.com/2024/01/11/playing-with-fire-how-we-executed-a-critical-supply-chain-attack-on-pytorch/.

[111] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. 2021. Invisible Probe: Timing Attacks with PCIe Congestion Side-channel. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[112] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. 2020. sRDMA: Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access. In *Proceedings of the 2020 USENIX Annual Technical Conference (ATC)*.

[113] Alpa Team. 2024. Alpa: Training and Serving Large-Scale Neural Networks with Auto Parallelization. https://github.com/alpa-projects/alpa.

[114] Google Brain Team. 2024. TensorFlow: An Open Source Machine Learning Framework. https://www.tensorflow.org/

[115] Uber Engineering Team. 2024. Horovod: Distributed Deep Learning Training Framework for TensorFlow, Keras, PyTorch, and Apache MXNet. https://github.com/horovod/horovod

[116] The kernel development community. [n. d.]. Sequence counters and sequential locks. https://docs.kernel.org/locking/seqlock.html.

[117] The PyTorch Foundation. 2024. PyTorch. https://pytorch.org.

[118] Thomas Trouchkine, Guillaume Bouffard, and Jessy Clédière. 2019. Fault Injection Characterization on Modern CPUs: From the ISA to the Micro-architecture. In *IFIP International Conference on Information Security Theory and Practice*.

[119] United States Attorney's Office, Eastern District of California. 2023. Former Navy IT Manager Sentenced to Over 5 Years in Prison for Hacking Computer Database. https://www.justice.gov/usao-edca/pr/former-navy-it-manager-sentenced-over-5-years-prison-hacking-computer-database.

[120] Stephan Van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. 2024. SoK: SGX.Fail: How Stuff Gets eXposed. In *Proceedings of the 45th IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[121] S. VanDeBogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazieres. 2007. Labels and Event Processes in the Asbestos Operating System. *ACM Transactions on Computer Systems (TOCS)* 25, 4 (December 2007), 11:1–43.

[122] Stavros Volos, Cédric Fournet, Jana Hofmann, Boris Köpf, and Oleksii Oleksenko. 2024. Principled Microarchitectural Isolation on Cloud CPUs. In *Proceedings of the 31st ACM Conference on Computer and Communications Security (CCS)*. Salt Lake City, UT.

[123] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. 1993. Efficient Software-Based Fault Isolation. *ACM SIGOPS Operating Systems Review* 27, 5 (Dec. 1993), 203–216.

[124] Han Wang, Linfeng Zhang, Jiequn Han, et al. 2018. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications* 228 (2018), 178–184.

[125] Robert Watson, Wayne Morrison, Chris Vance, and Brian Feldman. 2003. The TrustedBSD MAC Framework: Extensible Kernel Access Control for FreeBSD 5.0. In *Proceedings of the 2003 USENIX Annual Technical Conference*.

[126] Roy Weiss, Daniel Ayzenshteyn, and Yisroel Mirsky. 2024. What Was Your Prompt? A Remote Keylogging Attack on AI Assistants. In *Proceedings of the 33rd USENIX Security Symposium (Security)*. Philadelphia, PA.

[127] Charles V Wright, Scott E Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[128] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. 2022. Bedrock: Programmable Network Support for Secure RDMA Systems. In *Proceedings of the 31st USENIX Security Symposium (Security)*. Boston, MA.

[129] Jiarong Xing, Qiao Kang, and Ang Chen. 2020. NetWarden: Mitigating Network Covert Channels while Preserving Performance. In *Proceedings of the 29th USENIX Security Symposium (Security)*. Boston, MA.

[130] Wenjie Xiong and Jakub Szefer. 2021. Survey of Transient Execution Attacks and Their Mitigations. *ACM Computing Surveys (CSUR)* 54, 3 (May 2021), 1–36.

[131] Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar. 2009. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (Oakland)*. Oakland, CA.

[132] Shui Yu, Wanlei Zhou, Robin Doss, and Weijia Jia. 2010. Traceback of DDoS Attacks using Entropy Variations. *IEEE Transactions on Parallel and Distributed Systems* 22, 3 (2010), 412–425.

[133] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. 2006. Making Information Flow Explicit in HiStar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Seattle, WA.

[134] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. 2008. Securing Distributed Systems with Information Flow Control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. San Francisco, CA.

[135] Yan Zhai, Lichao Yin, Jeffrey S Chase, Thomas Ristenpart, and Michael M Swift. 2016. CQSTR: Securing Cross-Tenant Applications with Cloud Containers. In *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*.

[136] Chuqi Zhang, Rahul Priolkar, Yuancheng Jiang, Yuan Xiao, Mona Vij, Zhenkai Liang, and Adil Ahmad. 2025. Erebor: A Drop-In Sandbox Solution for Private Data Processing in Untrusted Confidential Virtual Machines. In *Proceedings of the 20th European Conference on Computer Systems (EuroSys)*. Rotterdam, The Netherlands.

[137] Duo Zhang, Xinzijian Liu, Xiangyu Zhang, Chengqian Zhang, Chun Cai, Hangrui Bi, Yiming Du, Xuejian Qin, Anyang Peng, Jiameng Huang, et al. 2024. DPA-2: a large atomic model as a multi-task learner. *npj Computational Materials* 10, 1 (2024), 293.

[138] Xiaokuan Zhang, Jihun Hamm, Michael K Reiter, and Yinqian Zhang. 2019. Statistical Privacy for Streaming Traffic. In *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[139] Yanli Zhao, Rohan Varma, Chien-Chin Huang, Shen Li, Min Xu, and Alban Desmaison. 2022. Introducing PyTorch Fully Sharded Data Parallel (FSDP) API. https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/.

[140] Ziqiao Zhou, Yizhou Shan, Weidong Cui, Xinyang Ge, Marcus Peinado, and Andrew Baumann. 2023. Core slicing: closing the gap between leaky confidential VMs and bare-metal cloud. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Boston, MA.

[141] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, and Dan Meng. 2020. Enabling Rack-Scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[142] Andrew D Zonenberg, Antony Moor, Daniel Slone, Lain Agan, and Mario Cop. 2025. Extraction of Secrets from 40nm CMOS Gate Dielectric Breakdown Antifuses by FIB Passive Voltage Contrast. In *Proceedings of the 19th USENIX Workshop on Offensive Technologies (WOOT)*.