GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs

Shixuan Zhao
The Ohio State University
Columbus, OH, USA
zhao.3289@buckeyemail.osu.edu

Zhongshu Gu
IBM Research
Yorktown Heights, NY, USA
zgu@us.ibm.com

Salman Ahmed
IBM Research
Yorktown Heights, NY, USA
sahmed@ibm.com

Enriquillo Valdez
IBM Research
Yorktown Heights, NY, USA
rvaldez@us.ibm.com

Hani Jamjoom
IBM Research
Yorktown Heights, NY, USA
jamjoom@us.ibm.com

Zhiqiang Lin
The Ohio State University
Columbus, OH, USA
zlin@cse.ohio-state.edu

Abstract

Confidential collaborative machine learning (ML) enables multiple mutually distrusted data holders to jointly train an ML model while preserving the confidentiality of their private datasets due to regulatory or competitive reasons. However, existing works need frequent data and model exchanges during training via slower conventional links. They face increasing challenges due to the exponentially growing sizes of models and datasets in modern training workloads like large language models (LLMs), resulting in prohibitively high communication costs. In this paper, we propose a novel mechanism called GPU Travelling that leverages recently emerged confidential GPUs. With our rigorous design, the GPU can securely travel to the specific data holder to load the dataset directly into the GPU's protected memory and then return for training, eliminating the need for data transmission while ensuring confidentiality up to a data-centre level. We developed a prototype using Intel TDX and NVIDIA H100 and evaluated its performance on llm.c, a CUDA-based LLM training project, and demonstrated the performance and feasibility while maintaining strong security guarantees. The results showed at least 4x speed improvement when transmitting a 512 MiB dataset chunk versus conventional transmission.

CCS Concepts

• **Security and privacy** → **Systems security**; • **Computing methodologies** → *Distributed artificial intelligence*; • **Computer systems organization** → **Distributed architectures**.

Keywords

TEE; Collaborative ML; Confidential Computing

ACM Reference Format:
Shixuan Zhao, Zhongshu Gu, Salman Ahmed, Enriquillo Valdez, Hani Jamjoom, and Zhiqiang Lin. 2025. GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs. 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/10.1145/3719027.3765029

CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765029

1 Introduction

With the continued advancement of Machine Learning (ML) systems, data security and confidentiality have become critical [41, 48]. This challenge is notably amplified in collaborative ML computations that work by cooperation between multiple mutually distrusting data holders. Under this context, participating data holders are often reluctant or even forbidden to share sensitive datasets due to privacy [53], regulatory [35], copyright [18], or competitive advantage.

Confidential computing has emerged as a viable and cost-effective solution to offer confidential ML by providing a trusted execution environment (TEE) that ensures data confidentiality and integrity [57, 58]. It encapsulates sensitive data and computation within a protected environment while also enabling remote attestation to verify the authenticity and integrity of the computing platform. For traditional CPU-bound workloads, hardware TEEs like AMD SEV [31] and Intel TDX [8, 26] offer confidential virtual machines (CVMs) to protect them and are widely available in commercial cloud platforms [1, 4, 19, 25]. The recent introduction of NVIDIA Confidential GPUs extends the trust boundary beyond CPUs, making confidential ML practical when combined with a CVM [39]. By allowing computations to be executed within a GPU TEE, this technology significantly enhances security guarantees for running AI workloads.

However, we find currently there are three major problems when using confidential computing with collaborative ML:

1. Scalability in Distributed Collaborative Training with Large Models. Modern ML models, particularly Large Language Models (LLMs), are characterised by their gigantic sizes, with parameter counts reaching over 600 billion [44]. Conventional distributed collaborative training schemes, such as federated learning (FL) and peer-to-peer (P2P) learning, require frequent aggregation and redistribution of locally trained models or updates, causing substantial communication overheads, and can be worsened when encryption is involved. Furthermore, these schemes are fundamentally limited by the requirement that each participant possess sufficient local computational resources (GPU compute power and memory) to train the entire model on its own. This requirement often becomes prohibitively demanding, or even infeasible, given the scale of modern large models [10].

2. Secure Dataset Transmission Overheads in Centralised Collaborative Training with Large Datasets. For centralised training, the training datasets have to be transmitted to the centralised server to be used in training. In large scale ML workloads like LLM, datasets can be massive [15] and the transmission cost over conventional links is high. When dataset confidentiality is required, datasets must also be encrypted and sealed when leaving the data holders and transmitted in ciphertext over untrusted channels. Upon arrival at the destination training server, it must be authenticated and decrypted before being used for training. Given the sheer volume of data involved in today's ML workloads, as shown in §7, the transmission cost using a regular encrypted channel like TLS/SSL is prohibitively high and can add a considerable amount of time and cost to the overall training.

3. Current Confidential GPUs Are Not Optimised for Collaborative ML. Confidential GPUs are currently designed for a single user usage, meaning that they do not trivially work in confidential collaborative ML in an efficient manner. A confidential GPU can only be assigned to a single CVM at a time and requires a full reset whenever it is detached from one CVM and reassigned to another [40]. While this design offers robust protection in a single user setting, it has no special optimisation for the need from confidential collaborative ML. Therefore, the collaborative ML has to be done via conventional methods like FL and centralised training discussed above even with a confidential GPU. This means that the scalability remains an issue, and the massive communications incurred still need to be carried through other slower conventional channels like TLS/SSL.

To summarise, the key observation is that **current generic confidential collaborative training techniques suffer from significant overheads due to transmitting a large amount of data over slower encrypted links.** Ideally, when we look at how the training process works, the model should stay on the GPU and the data should only be loaded into the GPU via faster links like PCIe. This is the most efficient approach and aligns with what is typically done in standard, non-collaborative training. Naturally, we came up with the following research question: *If all the problems come from transmitting large data over slower encrypted links for collaboration and confidentiality,* ***can we eliminate it while still allowing the confidential collaboration?***

> *Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway* [49].
> ——Andrew S. Tanenbaum

The historical quote by Andrew S. Tanenbaum inspired us that we can enable the GPU to *travel* to data holders to collect the data using high-speed PCIe links and then come back for training, in contrast to conventional methods of transmitting the datasets/models between training participants. We further designed the system to work with the GPU's confidential computing mechanism. The datasets' confidentiality can be protected by performing dataset clean up and key rotation so that when the GPU travels to a different data holder, there is no possible leakage on the datasets.

This generic approach can be easily adopted for most training workloads and can accelerate to the scale up to data-centre level confidential collaborative ML on its own. It eliminates the need to transmit large datasets or models over slow network connections, significantly improving training efficiency while preserving the confidentiality of the datasets. It can also be combined with existing techniques like FL to achieve an even wider distribution.

The primary challenge of this GPU Travelling design is that current confidential GPU architectures erase all data and contexts when being reassigned between CVMs. To address this, we must be able to '*hot* reassign' confidential GPU to CVMs on the fly. Even more challenging, to make this mechanism practical, we must achieve this without modifying any proprietary part. But GPUs, particularly NVIDIA GPUs, consist mostly proprietary components with the exception of the driver [38]. We introduce two key innovations to achieve this:

- **PCIe MUX.** We developed a virtual PCIe multiplexer (MUX) that is capable of hot re-routing the physical PCIe connection to a target CVM so that the GPU does not have to be reset to be physically reassigned to CVMs.
- **Confidential Context Travelling.** We heavily modified NVIDIA H100's confidential computing logics in the driver so that the necessary driver contexts to communicate with the GPU can be exported/imported, allowing a GPU to retain its context without affecting functionality or compromising security.

Our experimental results demonstrate that GPU Travelling can significantly reduce data transmission overhead while our security analysis shows that the confidentiality of the datasets is maintained. The proposed approach achieves substantial improvements in training performance for large-scale AI models compared to existing confidential computing schemes and can improve the dataset provision by at least 4.5x to over 1,100x for a 512 MiB dataset buffer depending on the available bandwidth. Our evaluation on a real-world application llm.c also showed considerable performance improvement as shown in §7.

Contributions. We make the following key contributions:

- We introduced ***GPU Travelling***, a novel mechanism designed to mitigate the substantial communication overheads inherent in confidential collaborative training scenarios.
- We developed and presented key techniques enabling the secure travelling of a confidential GPU across GPU-Travelling-aware CVMs while preserving the confidentiality of the datasets and security properties.
- We demonstrated the practical feasibility and significant performance improvement of GPU Travelling via prototype implementation and comprehensive empirical evaluation.

2 Background

2.1 Confidential Collaborative ML

Under confidential collaborative machine learning, multiple mutually distrusted *data holders* want to train a single model out of their private and confidential datasets. A wide array of scenarios can benefit from this scheme. For example, an LLM can be trained with copyrighted materials from different presses that do not wish to leak the copyrighted materials; A cancer detection model can be trained from confidential and legally-protected medical records from different hospitals. While there are many existing works trying to tackle the problem, they primarily fall into two categories: transferring ***model*** or transferring ***dataset***.

Transferring Model. The most prominent method of model transferring for confidential collaborative ML is Federated Learning (FL) [32]. In FL, a centralised node distributes a base model to each data holder who then trains the model with its private dataset using its own computing resources locally. After all data holders have trained the model, the gradients or the model parameters are uploaded to the centralised node where these gradients or parameters are consolidated into one model. The procedure can be repeated for the desired epochs as needed. There are two problems of FL that we can conclude from the procedure. One is that the size of the differences of gradients can be quite large once the model is large. There are methods to compress the differences for specific kinds of workloads like [56] but is overall challenging to be applied to general workloads. The second problem is very difficult to avoid, which is the requirement that each data holder must have enough computing resources like GPUs to train on their own datasets. This in many cases can be impractical [10].

Transferring Dataset. Another way to achieve confidential collaborative ML is to transfer the datasets to a trusted centralised training node. The training node can be protected and attested using TEE to ensure the confidentiality as in [6, 55] or deprivileged with multi-party computation and homomorphic encryption as in [9]. To ensure the confidentiality of the datasets, the transmission must be encrypted to prevent leakage. However, datasets used in modern training workloads have grown significantly. This means that transferring the dataset can cause severe performance degradation, particularly with additional overheads caused by encryption.

2.2 Confidential VMs

VM-based TEEs are a kind of hardware TEEs that protect an entire VM against privileged attackers like the hypervisor from stealing or even compromising the VM. Such a protected VM is called as a Confidential VM (CVM). The mechanism in the hardware encrypts the memory of the CVM in the MMU [31], ensuring exclusive use of memory pages [2], protected CPU registers [30], and some other protection mechanisms like restricted interrupt delivery [2]. Inside a CVM, an enlightened kernel is provided to handle the TEE features of the hardware while leaving the user space fully compatible with existing workloads and therefore retains great compatibility [54]. There are currently two dominant VM-based TEEs on the x86 platform: Intel TDX [8, 26] and AMD SEV [31]. Despite certain implementation differences, they share great similarities in terms of protection as well as operation and are both widely supported. In CVMs, conventional PCIe devices including GPUs and network cards that connect to the CVM are untrusted. Hypervisors can intercept or manipulate the communication if no extra protection mechanism is implemented in the devices [39, 59].

2.3 NVIDIA Confidential Computing

In CVM's threat model, external devices like GPUs are out of the secure world and are neither trusted nor protected. However, in modern ML workloads, GPUs are indispensable to achieve practical performance. To solve this dilemma, NVIDIA proposed NVIDIA Confidential Computing (CC) to create a GPU TEE that works with CVMs and debuted it on NVIDIA H100 [39].

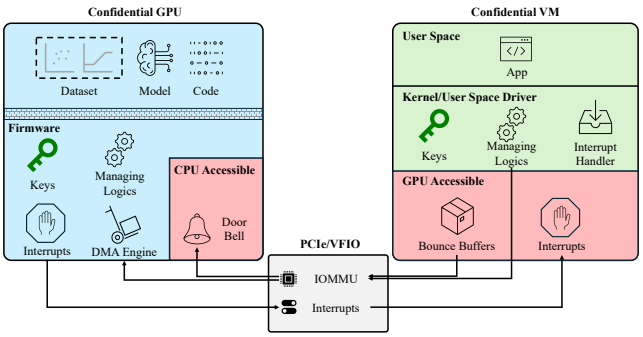

Figure 1: The TEE architecture of NVIDIA H100

In NVIDIA CC, a GPU's memory is protected against privileged attackers from accessing from the PCIe bus with a firewall mechanism [20]. In addition, the communication between a GPU and a CVM is also encrypted. The architecture of NVIDIA H100's CC implementation is illustrated in Figure 1. During the initialisation of the GPU, a Security Protocols and Data Models (SPDM) [12] session is created to attest the GPU and to build an asymmetric encrypted communication path over the unprotected PCIe link. This allows the GPU and the CVM's driver to negotiate symmetric keys for further communications. Multiple unprotected *bounce buffers* are allocated on the CVM side for different functionalities. When the GPU or the CVM needs to communicate with each other, the content will be encrypted using the specific functionality's symmetric key and then placed into the designated bounce buffer. The GPU can use PCIe DMA to read/write the bounce buffer with cipher text [39] and then work in its protected memory on the plain text. This means that privileged attackers like a hypervisor can no longer read or modify the communication and the workloads relying on GPUs can be protected.

3 Overview

As we have discussed in §2.1 that existing confidential collaborative training techniques requires the transferring of the model or the dataset. Methods that transfer models like federated learning requires each data holder to have its self-owned computing resources; Both methods impose significant communication costs on large model/dataset transmission. As shown in our evaluation in §7, the transmission cost over TLS/SSL is significant and can take a considerable portion of the entire training process depending on the bandwidth.

The GPU Travelling mechanism is designed to eliminate the need of transferring massive amount of data over slower links in confidential collaborative ML when all data holders can have a direct PCIe link to a single group of confidential GPUs. This, however, requires a rigorous and non-trivial design to achieve both the performance improvement while maintaining the confidentiality. In this section, we discuss the threat model and the design goal of GPU Travelling.

3.1 Assumptions and Threat Model

In our target scenario, we consider a cloud-based training setting that each data holder operates a CVM that is only accessible to itself

exclusively to securely host and provision their datasets. While being distributed, data holders need to have direct PCIe links to a single GPU, either locally on the same server or through intra-data-centre PCIe fabrics. They try to train a single model out of their datasets but want to keep the datasets confidential to each other.

Data holders mutually distrust each other, and the platform operator, responsible for providing servers and GPU resources, is also untrusted. We consider two primary types of adversaries, ensuring dataset confidentiality even in scenarios where adversaries collude.

- **Internal Adversaries: Malicious Data Holders.** Malicious data holders aim to illegitimately access or steal datasets belonging to benign data holders. Although isolation mechanisms enforced by the TEE prevent direct breaches, these adversaries might exploit our provisioning mechanisms to indirectly infer or even access confidential datasets.
- **External Adversaries: Malicious Platform Operator.** As the platform operator controls hardware and hypervisor layers, it is inherently untrusted. While direct access to the CVM and GPU TEE is restricted, the platform operator may exploit hypervisor-level capabilities, such as manipulating PCIe connections or assigning the confidential GPU to incorrect CVMs.
- **Collusion Attacks.** We explicitly consider scenarios where internal and external adversaries collude, thereby combining their individual capabilities to launch more sophisticated attacks.

We assume that Denial-of-Service (DoS) attacks, side-channel attacks and attacks to the hardware to be out the scope of this work. Note that inferencing dataset information through model difference (e.g., model inversion) is also considered orthogonal to this research with existing works mitigating the issue [13, 33, 50, 52].

3.2 Design Goals

To solve the problem of high communication costs involved in confidential collaborative training, our proposed mechanism must be able to achieve the following goals:

- **G1: Eliminating Slow Transmission of Large Dataset or Model.** Our mechanism must completely eliminate transmitting either datasets or the model through slow connections like Ethernet. The only transmission of datasets or the model should be the GPU loading procedure through the direct PCIe link.
- **G2: No Impact to Training Results.** Our mechanism must not disturb other processes of the training. It must not introduce any technical impact to the resulting model.
- **G3: Imposing Low Mechanism Overheads.** The overheads caused by the mechanism itself must be low so that it does not introduce extra burden when compared to a conventional solution that sends over slower connections like Ethernet.
- **G4: Preserving Confidentiality of Datasets.** When using our mechanism, the confidentiality of each data holder's datasets must be preserved against malicious attackers discussed in §3.1.
- **G5: No Change in Proprietary Components.** While typically we can obtain an open source GPU driver (e.g., NVIDIA's Open GPU Kernel Modules [38]), the rest of the computing architecture remains proprietary (e.g., firmware, user space runtime like CUDA). Our mechanism must not require any changes in these proprietary components.

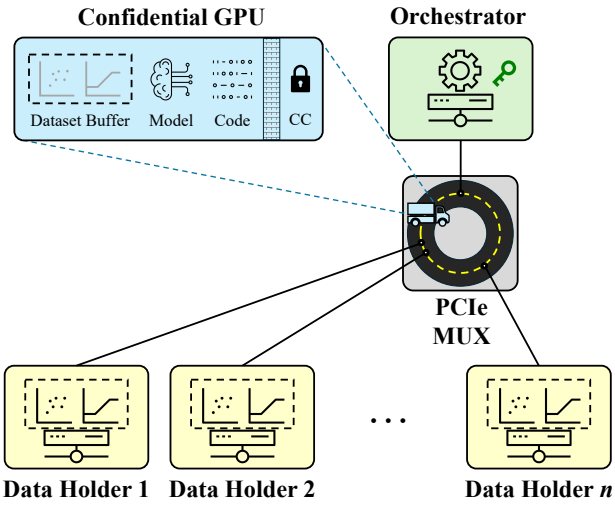

Figure 2: System overview of Travelling GPU.

3.3 System Overview

As discussed in §1, we envision the GPU to operate like a truck that *travels* to each data holder to collect data and returns for training. We call this concept *GPU Travelling*. While the idea can be described in a few words, the design is non-trivial in order to satisfy our goals discussed in §3.2.

The system overview of GPU Travelling is illustrated in Figure 2. As discussed in §3.1, we consider that each data holder (§4.2) operates a corresponding CVM that is only accessible by the data holder itself exclusively so datasets can be securely provisioned into. These data holders can inspect, verify and agree on an autonomous CVM we called as the *orchestrator* (§4.2), which conducts the entire training procedure on its own and denies any access from any data holder once set up (§4.1). Both the orchestrator and data holder CVMs have a direct PCIe link to a single GPU where the PCIe link can either stay locally on the same server or be connected via an external PCIe fabric that can allow a distribution up to a data-centre level.

The workflow mostly remains the same as conventional single-node training on the orchestrator except for the dataset loading. In conventional single-node training, the dataset is located on the training node and is directly copied into the GPU. In our system, instead, we perform a secure switching of the GPU to the corresponding data holder. Since we use a confidential GPU, there are 2 layers of switching needed as illustrated in Figure 1. We first need to do a 'hard' switch on the physical layer by re-routing the GPU's PCIe link to the data holder's CVM (§4.3) and then do a 'soft' switch on the security layer that provisions the driver contexts including the communication keys to the data holder's CVM (§4.4). Note that the GPU remains in confidential mode and will only respond to those who have valid keys. The data holder CVM can then access the GPU and copy its datasets into it. After datasets are copied, the data holder's CVM will similarly perform the switches on the two layers back to the orchestrator to continue the training procedure. A detailed step-by-step workflow is presented in §4.6 after presenting the components in GPU Travelling.

By doing so, we satisfy **G1** that neither the datasets nor the model needs to be transmitted via slower connections as the datasets are provisioned directly into the GPU and the model is held on the GPU. It also satisfies **G2** since the only difference when comparing to a conventional single-node training is how the same piece of dataset is provisioned into the GPU. To achieve **G3**, the design and the engineering of the mechanism must minimise the overheads and provide a significant performance improvement over the conventional way, which will be discussed in details in §4.3 and §4.4. Since there are multiple mutually distrusted data holders that may get access to the GPU, careful designs must be made in the security layer to achieve **G4** which will be discussed in §4.4. The most challenging part comes from **G5** that we must make our mechanism generic and practical enough so that it does not require changes in the proprietary components. These challenges will be brought up in §4 but will be further discussed in details in §5.

4 Design

4.1 Bootstrapping

The bootstrapping procedure of GPU Travelling imposes requirements on the order of CVM booting and the GPU context each CVM needs to hold before booting the next one.

Boot Order. As each CVM boots, the contexts on the GPU created by the previous CVM is wiped and only the last-booted CVM has the full control and context for the GPU. Therefore, while the order of how data holder CVMs boot can be arbitrary, the orchestrator CVM must be booted as the last one.

Context Setup on Data Holders. For each data holder's CVM, it must perform a GPU setup to initialise the driver on its end before booting the next CVM. The context will become invalid after the next CVM boots but can be restored later as to be discussed in §4.4. This includes the interrupt handlers, PCIe BAR mappings, DMA mappings for necessary infrastructures like the bounce buffer and other necessary driver and user space contexts. The data holder should be set up to a point where it is ready to copy the datasets right after taking the GPU.

4.2 CVM Roles

In our design, there are two types of CVMs involved: a single *orchestrator* and multiple *data holders*. We describe them in details.

Orchestrator. The orchestrator is a trusted, autonomous CVM that runs on its own. It hosts the training code and logics in it, and conducts the training process. It can be adapted from a conventional training node by changing the routines for dataset loading. Note that because we use a confidential GPU, the GPU itself must also be attested and then be set up with the confidential computing context (e.g., communication keys).

Before loading any dataset, the orchestrator first needs to allocate a dataset buffer in the GPU's VRAM for the data holder to load the dataset. It can then perform a switching to the data holder then supply the dataset buffer's address and size to request a loading. When the data holder has loaded the dataset into the GPU and switched the GPU back to the orchestrator, the orchestrator can perform the training just like the conventional way. Note that the request to the data holder must be encrypted and additional cleaning must be done to ensure the security, which will be further discussed in §4.4.

The orchestrator itself does not contain any data and therefore can be inspected and verified by every data holder before the deployment to ensure that no malicious logic is present. It is the only globally trusted software component in the system other than the firmware of the hardware. Note that the training application in the orchestrator CVM can be written using proprietary SDKs like CUDA. It has full GPU contexts for both the driver and the user space.

Data Holder. Each data holder operates a CVM, in which they can securely hold their datasets. These data holder CVMs are able to physically connect to the GPU via PCIe link. Data holder CVMs are mutually distrusted and each of them will only build a secure and encrypted connection with the orchestrator after attesting the identity and integrity of the orchestrator CVM that is globally trusted by every data holder. Since the orchestrator CVM is inspected to make sure it will attest the GPU, a data holder CVM can also trust the GPU that is switched to it from the orchestrator. This guarantees that for each dataset loading request comes from the trusted orchestrator and the dataset is only loaded to a trusted confidential GPU.

The workflow on the data holder side is essentially a server. Once the GPU is switched to the data holder CVM with the dataset buffer's address and size supplied, it copies a portion of the dataset that fits into the buffer's size into the GPU and returns the GPU to the orchestrator.

For data holders, if the orchestrator's training application is written with proprietary SDKs like CUDA, to satisfy our design goal **G5**, the data holder server can no longer retain GPU context in the user space. However, we can provide an API in the driver to copy the dataset into the GPU so the server only has to call that API to achieve the copying.

4.3 PCIe MUX: The Physical Layer

One of the key technical challenge of the GPU Travelling is to pass through a single GPU on the physical PCIe layer into different CVMs on the fly. A PCIe link consists interrupts and DMA mappings and is considered as an untrusted link in the confidential GPU as illustrated in Figure 1. We propose a component called a PCIe MUX in the untrusted hypervisor to achieve this because it functions like a multiplexer (MUX).

Interrupts. Traditionally, a PCIe device uses a single line interrupt, which is still the default behaviour of the device after a reset. However, modern PCIe devices like NVIDIA's H100 support Message Signaled Interrupts (MSI) and MSI-X that is faster and supports more interrupts. After the driver is loaded and the GPU is set up, MSI-X will be enabled. Therefore, our mechanism must support the switching of these standards. We register each CVM's interrupt routing endpoints. When switching the interrupts to a specific CVM, we connect the GPU's interrupt lines with the corresponding endpoints so the interrupts will then be routed to that CVM.

DMA Mappings. To build the address space used by the GPU, IOMMU is used to hold the mapping from the physical memory to guest physical address as the IO Virtual Address (IOVA), which

the GPU uses to access the corresponding memory locations. The mapping includes unencrypted DMA zones of the CVM as well as the MMIO zone of the GPU itself.

To achieve the switching, we register each CVM's mappings just like the interrupts. When switching, we remove the previous CVM's mappings and replay the saved mappings of the target CVM.

By switching the interrupts and the DMA mappings, the confidential GPU's physical link can be switched between multiple CVMs. Note that the PCIe MUX described above is a software component in the host kernel. One may combine it with an external PCIe switch/fabric to achieve switching the GPU to other physically distributed servers. Interestingly, from the perspective of the GPU, it has no knowledge that it is being switched. However, without the keys and the contexts, even if the GPU is physically connected to a CVM, the CVM cannot communicate with the GPU.

4.4 Confidential Context Travelling: The Security Layer

After the GPU is physically switched to a CVM, the CVM must also obtain the keys and other context information from the previous CVM to communicate with the GPU. This requires a rigorously designed setup and switching mechanism to ensure both functionality and security.

Context Setup. GPU drivers (e.g., NVIDIA's driver) in each CVM communicates with the GPU using buffers and door bell signals on the lowest level. On confidential GPUs, signals are done using PCIe register writes and interrupts while there is an extra layer called *bounce buffers* for staging ciphertexts in the CVM's DMA zone that are negotiated at the setup. We have already made sure the same MMIO for PCIe registers and interrupts used for the signals are connected as expected in the PCIe MUX. However, as we must not make changes in proprietary components, we cannot require the GPU firmware to have the ability to change the address of these predefined bounce buffers each time we switch. Therefore, the bounce buffer addresses must remain the same across different CVMs.

To do this, we require each CVM to reserve a small amount of RAM that is enough to fit in the buffers at a static physical address. Then, for each of the CVM, we can ensure that the bounce buffers are always allocated at the same address by modifying the memory allocator on the driver side so the GPU can communicate through the correct bounce buffers in any CVM.

Context Switching and Key Migration. When switching to another CVM, the necessary driver contexts must also be provisioned into the target CVM including message queue status, semaphore status, etc. Without these, the context in the target CVM's driver will be out of sync with the GPU's and the communication between the two may crash.

The most important part of the switching, which actually empowers the target CVM to talk with the GPU, is the key migration. Confidential GPUs like NVIDIA H100 uses AES-GCM and requires not only the key but also an Initial Vector (IV) to function as a nonce. This prevents replay attacks and also allows to tell the timing for a key rotation. The key-IV pair is assigned per-channel and therefore we only need to provision the channels we would use during the communication. Note that after migrating the key to the target CVM, the previous CVM can still decrypt the communication between the target CVM and the GPU as it has the key and the IV. In case it wants, it can even intercept the communication and pretend to be the target CVM to talk with the GPU with proper timing. We must introduce additional steps to prevent the data leakage in our design.

Security Clean Up. While the switching mechanism we discussed so far is symmetric between the orchestrator and the data holder, the trust relationship is not. As we discussed, the orchestrator is globally trusted but data holders are not. After the orchestrator has collected all the dataset from a data holder and finished training, it will move on to the next data holder. From the above discussion, we know that the previous CVM (here the previous data holder) may still retain the keys and IVs. If we use the same keys and IVs for the next data holder, the previous data holder may be able to decrypt the communication and steal the dataset. To prevent this, each time when switching to a different data holder, the orchestrator must scrub up any residual dataset information on the GPU and perform a key rotation. By doing this, the previous data holder's data is removed with no leakage risk and the keys are invalidated so the previous data holder no longer has the ability to decrypt future communications.

4.5 Chain of Trust

The chain of trust is built based on the hardware roots of trust for both the CPU and the GPU in two phases: GPU phase and CPU phase. When the orchestrator CVM is launched, it will first perform an attestation to the GPU. The GPU will use its hardware root of trust to establish the trust which is verifiable from the GPU's vendor (e.g., NVIDIA). This ensures that the GPU is genuine and trusted. The encrypted channel between the GPU and the orchestrator is also built at this phase. In the next phase, data holders join the training cohort and connects with the orchestrator. In our threat model, the orchestrator is fully trusted and its code can be inspected by the data holders. Therefore, by verifying the orchestrator's deployment using CVM attestation on the CPU side, a data holder can establish trust on both the orchestrator and the CPU. Since the orchestrator will also attest the GPU, the data holder can now also trust the GPU. This forms the chain of trust in GPU Travelling.

In an asynchronous environment, a data holder with a properly setup can decide to join later or leave at any time. To join, it may perform its attestation to the orchestrator CVM until it's willing to join without a problem. Since the data holders are symmetric, the cohort can perform the training as long as there is at least one online data holder without relying on any specific data holder. To leave, it may just inform the orchestrator its intention to leave and stop providing data. As long as its driver context is valid, it may rejoin at any future moment with either previously established trust or perform a new attestation.

4.6 Workflow

With the discussion above, we present the complete step-by-step workflow of our GPU Travelling mechanism and illustrate it in Figure 3. The process starts from ❶ security clean up so that no previous dataset buffer will be leaked. The orchestrator training

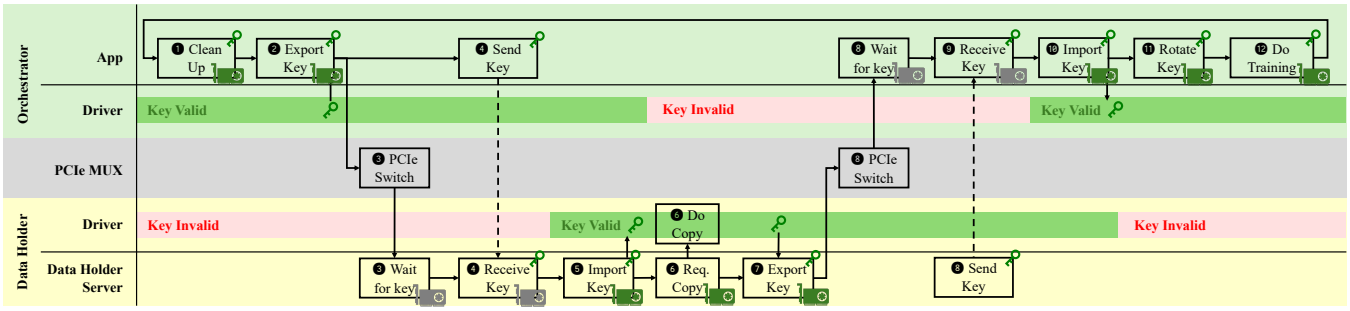

Figure 3: Workflow of GPU Travelling. Note that a GPU icon means the GPU is on the specific CVM. A green coloured GPU means the CVM has the key to access the GPU. A grey coloured GPU means the GPU is physically connected to the CVM but without key.

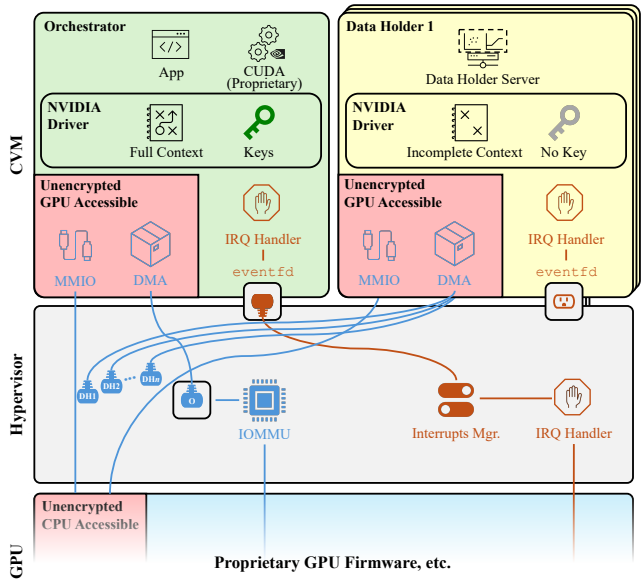

Figure 4: The system architecture of Travelling GPU

application ❷ exports the keys and contexts and then initialise a ❸ PCIe switch to the data holder. It then ❺ sends the keys and contexts so that the data holder can import the keys and contexts. After this, the data holder server, as described in §4.2, needs to request the driver to ❻ copy the dataset into the GPU. Once copied, the data holder then ❼ exports the keys and contexts back, ❽ initiates the PCIe switching to the orchestrator and then ❾ sends the keys and contexts back to the orchestrator. After ❿ importing the keys and contexts, the orchestrator can immediately ⓫ rotate the keys so the data holder no longer has access to the GPU even with a PCIe link. The orchestrator ⓬ performs the training on the dataset and loops back to the beginning.

5 Implementation

To demonstrate the feasibility of the Travelling GPU mechanism, we implemented a working system with an NVIDIA H100 on an Intel TDX platform. The code is based on the open source version of NVIDIA's driver version 560.35.03 for the driver, and Linux 6.12.0-rc1 with TDX patch 2024-11-08-build-442 on the host side for the PCIe MUX. Our modifications on the code consist of 4,746 LoC and have been made available along with tools and libraries for reproduction at https://zenodo.org/records/16899384. The system architecture with detailed block diagrams is illustrated in Figure 4.

5.1 VFIO-Based PCIe MUX

We implemented our PCIe MUX with a modified Virtual Function I/O (VFIO) driver in the host-side Linux kernel. Conventionally, VFIO would assign a single device to be bound with only a single VM and would prevent the device from being assigned to another VM until the previous VM releases the device. We modified the driver so that it will allow assigning to multiple VMs by saving and overwriting the previous VM's configurations each time when being assigned to a new VM. As discussed in §4.3, we need to handle both the interrupts and the DMA mappings. We discuss the implementation details here.

Interrupts. The interrupt handling in VFIO of a VM consists of two endpoints for conventional interrupts: the hypervisor side IRQ handler and the `eventfd` of the QEMU passed into the VM. An interrupt is first setup with an IRQ handler on the hypervisor side, which will signal the `eventfd` created by the CVM's QEMU's interrupt subsystem. The QEMU will do VFS polling on the `eventfd` and once the `eventfd` is signalled, it will inject the interrupt to the VM with KVM. In modern standards like MSI-X that is switched to by H100 once the GPU driver is loaded, the interrupt can also go through a mechanism called KVM bypass, in which the hardware will inject the interrupt directly into the VM without the need of the VFS polling process.

To perform the switching, we keep the hypervisor side IRQ handler unchanged throughout the lifecycle but replace the `eventfd` and the bypass end points. This means that whenever a VM joins/updates the interrupt, we only change/save the `eventfd` and bypass information and swap them up when switching. The VM side end points are like sockets on the VMs while our connection

is like plugging a connector into the corresponding VM with the other end connecting to the host's IRQ allocation.

DMA Mappings. The DMA mappings are per-VM-specific and is done using IOMMU so that the GPU can and can only access these mapped DMA memory as needed. As discussed in §4.3, the GPU accesses through IOVAs that are essentially the guest VM's physical addresses. Therefore, each IOMMU mapping entry maps a guest physical address to a host physical address.

Just like the interrupt, we save the DMA mapping information whenever a VM joins/updates its DMA mapping. When switching, we first remove the previous VM's mappings (but keep the saved mappings) and replay the saved mappings of the target VM. This, however, differs from the interrupt as each set of DMA mappings is specific for each VM so it works like each VM has a connector that can be plugged into a single IOMMU socket as shown in Figure 4.

The GPU's BAR mappings are special DMA mappings that, instead of corresponding to physical RAM in the system, maps to the MMIO region of the GPU. They remains the same across different VMs and therefore we keep them when switching to improve the performance.

5.2 Key Migration

In NVIDIA H100's confidential computing, communication with the GPU including copying between the GPU and CPU is done through encrypted channels that are bound to a mechanism called Copy Engines (CEs). While all CEs are functionality wise the same, during the setup, each CE will be designated to back a single channel pool for a specific purpose. Each channel pool can have multiple channels in it. Although channels on the same CE uses the same set of keys, each channel maintains its own IVs used by AES encryption.

To allow a CVM to talk with the GPU, we must have valid keys as well as other cryptographic contexts like the IVs. Note that NVIDIA's driver contains multiple components besides the basic GPU driver. In particular, data copying between the GPU and the CPU on computation is done through the NVIDIA Unified Virtual Memory (UVM) kernel module (`nvidia-uvm.ko`). We modified both the basic GPU driver and the UVM driver to enable exporting and importing two pieces of the keys and contexts from both drivers. We also provide a user space library with APIs to do this at once and pack them into a single piece of data for easier operations.

5.3 Driver Contexts Migration

Besides the cryptographic keys we need to migrate, we also need to migrate the necessary driver contexts used during the CPU-GPU copying. There are multiple contexts on different layers and we discuss the context we need to migrate of each layer by describing the CPU-GPU copying workflow.

- **CPU-GPU Channel.** The CPU-GPU copying starts with the request being packed into a '*push*' data for the CPU to GPU channel. There is a semaphore linked to the channel to indicate the progress called a tracking semaphore. In this step, we need to know the dataset buffer's address on the GPU that is packed into the request as well as synchronizing the tracking semaphore.
- **Work Launch Channel (WLC).** In NVIDIA confidential computing, there is no way to do direct push on a regular CE channel due to restriction imposed. To solve this problem, NVIDIA H100 uses a special type of channel called a Work Launch Channel (WLC) [20] to indirectly submit the push of other channels. For each WLC channel, it has a fixed-size preallocated buffer for its own push. The GPU knows this buffer's address during the setup. The CPU-GPU push is encrypted into a pre-allocated push buffer and then the push buffer's address is added into a WLC push. An ID generated by the GPU indicating which WLC is being used called a Work Submission Token is also included in the push. The WLC push is written into a ring buffer on the GPU's MMIO zone called a GPFIFO (presumably the name comes from General Purpose First-In-First-Out) with each GPFIFO entry being 64-bit for a fixed schedule. The ring buffer's usage is recorded in a put-get fashion with two offsets corresponding to put and get. Once GPFIFO entry is written, the Work Submission Token is written into the GPU's doorbell location and the GPU would be able to know which WLC submitted the push and read the corresponding GPFIFO entry, decrypt and do the WLC push in a FIFO order. It will read the WLC push and the decrypt the push buffer containing the CPU-GPU push to do the actual push payload. Each WLC also has its own tracking semaphore to indicate the progress of the push. While this process is relatively complex, most contexts involved do not need to be migrated at this step as the addresses are either provisioned each time or statically allocated. Only the Work Submission Token, GPFIFO entry's offset and the tracking semaphore need to be synced.
- **Launch Confirmation Indicator Channel (LCIC).** Once the WLC push is submitted and done, the GPFIFO's put offset must be updated. This is done by a specialised channel called LCIC [20]. For each WLC, there is a paired LCIC that is responsible for updating GPFIFO's put offset of the WLC. LCIC has an encrypted static buffer to receive the new put value from the GPU and also has a tracking semaphore for the progress but would not check until the next WLC push. However, as there can be multiple pushes during the dataset copying, we must still sync the tracking semaphore.
- **GSP RPC.** The communication with the GPU System Processor (GSP) is required to perform certain allocation and management tasks during the copying. This is done via a Remote Procedure Call (RPC) [20] mechanism with a message queue as the underlying infrastructure. The RPC has a static bounce buffer that is also known on the GPU side. RPC commands and data are encrypted into that buffer and then sent through the message queue. The message queue maintains a pair of Tx/Rx sequence numbers. We have to sync the sequence numbers so to allow the GSP RPC to work.

5.4 Statically Allocated Bounce Buffer

From above, we can see that there are many statically allocated bounce buffers that are determined during the setup so the GPU does not have to query each time for the addresses of them. While this simplifies the GPU's firmware implementation, it actually imposes challenges on our implementation. As discussed in §4.3, the IOMMU actually maps the guest physical address to the GPU. Therefore, when GPU accesses the VM's memory, it is actually accessing through physical addresses. We therefore have to guarantee that it allocates the same physical address each time for these buffers.

To achieve this, we first reserved a small amount of physical RAM for these buffers at boot time. Then, we implemented a special allocation facility for this reserved area of RAM. The RAM is divided into *slots* with each slot corresponding to a dedicated buffer. When allocating a specific buffer, the slot number is given so that it will always be at the same physical address.

Note that the GPU has its own MMU with its own virtual address space. When a GPU virtual address is used, it can be safely synced to other CVMs without needing to adjust the mapping as the GPU's access through the PCIe will be using the physical address that remains the same throughout the execution.

5.5 Data Holder Driver API For Copying

As discussed in §4.2, when proprietary GPU SDKs like CUDA are used, the user space GPU contexts cannot be migrated and the data holder server needs to use the driver to copy the dataset into the GPU. This is also achieved by modifying the NVIDIA UVM driver to add new APIs that perform the copying. In the UVM driver, there are already GPU memory copying routines that are used internally in the driver. We wrapped up the routines into `ioctl` calls so that the data holder server can use them through a pre-opened UVM `ioctl` character file by offering the dataset buffer's address.

6 Security Analysis

In this section, we discuss the security of the system under our threat model §3.1 to see how we are able to achieve the protection of the confidentiality of the datasets.

6.1 Malicious Data Holder

A critical threat in our design is a malicious data holder since it will have the chance to access the GPU and retain the keys afterwards when it is being asked to copy its dataset into the GPU. On its own, a malicious data holder only has access to the GPU when the GPU is switched to it. In this case, it has access and control to the entire GPU. We consider two types of attacks it might perform: Dumping the GPU memory and planting gadgets in the GPU.

Dumping the GPU Memory. As described in §4.4, a clean up to the previous data holder's dataset buffer is done before the GPU travels to the new data holder's CVM. Therefore, for the new data holder, all it can read from the GPU is the model. While certain global properties of the dataset used in training may be inferred from the difference of the model between two epochs (i.e., model inversion), this kind of attack is considered as orthogonal to this research with existing literatures describing mitigations on this [13, 33, 50, 52]. Furthermore, we argue that if there are multiple data holders, the ability to distinguish a single data holder's dataset's property is low.

Planting Gadgets in the GPU. A malicious data holder may also try to change the contents inside the GPU to leave potential gadgets. However, since every piece of the code (*kernel* in CUDA term) is loaded freshly each time, it is not possible to cause code to misbehave. A malicious data holder can also modify the model so that it can reflect the dataset more clearly. But as discussed above, it is orthogonal to this research.

6.2 Malicious Hypervisor

A malicious hypervisor in our system is completely out of the TCB. It is guarded against the CVM and confidential GPU but is in control of the PCIe MUX. This means that it has the ability to switch the GPU to any CVM it wants or even CVMs belonging to any data holder or the orchestrator. It may even split the DMA mapping and interrupts to different VMs.

However, as the PCIe MUX works on the physical link layer, even if the interrupts and the DMA mappings are swapped to a wrong VM, that VM will not have the keys for the communications and therefore cannot read/write to the GPU. If the interrupts and DMA mappings are split into different CVMs, it will only cause page faults or unexpected interrupts to the CVMs and then crash the GPU instead of leaking any dataset information, which is considered as a DoS attack that is out of the scope of this research.

6.3 Collusion

A more powerful attacker can be formed from the collusion of a malicious data holder and a malicious host. In this case, the malicious data holder temporarily has the access to the GPU and may retain the keys while the hypervisor has the ability to switch the GPU to any CVM. The threat must be contained even under this circumstance.

In our system, before the GPU travels to a CVM, the orchestrator clears the dataset buffer that contains the previous data holder's dataset. This means that when a malicious data holder gets the GPU, all it can do is to retain the key and then collude with the hypervisor after the GPU travels back to the orchestrator. To steal other data holder's dataset, the malicious data holder must at least wait for the GPU to travel to another *victim* data holder and then intercept the communication. There are two ways to intercept the communication and we discuss below on how they are prevented.

GPU Control with a Malicious Switch. This can be done by colluding with the hypervisor to force a GPU to switch to the malicious data holder. If the GPU is switched to the malicious data holder who has valid keys then the malicious data holder can control the GPU and directly read the dataset buffer at will. However, as the orchestrator rotates the key before the GPU travels to the victim data holder, the keys retained by the malicious data holder are no longer valid and cannot be used to communicate with the GPU any more.

A malicious data holder may also try to stop the GPU from rotating the key by asking the hypervisor to drop the key rotation command sent to the GPU. However, the commands are encrypted and even if the hypervisor was able to identify the command and drop it, since the command's completion result would never return to the orchestrator, the orchestrator will be stuck waiting and will not proceed.

Intercepting the Bounce Buffer. Since the bounce buffer transmitting the ciphertexts is located in the unencrypted DMA zone of the CVM, the hypervisor may read the content of the ciphertext and request a decryption from the malicious data holder. However, this can be similarly prevented due to the key rotation.

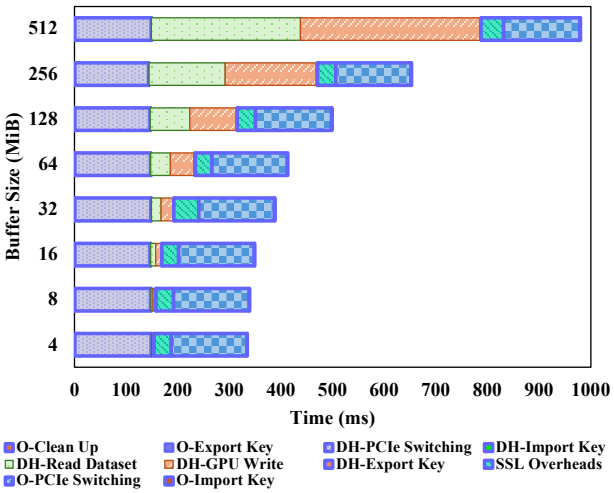

Figure 5: Breakdown of the GPU Travelling mechanism overheads. Steps with thick blue borders are those introduced by the GPU Travelling mechanism.

7 Evaluation

In this section, we present our evaluation of the system to demonstrate the improvement of the overheads of confidential collaborative training using the GPU Travelling mechanism. We designed our evaluation experiments to systematically quantify the our system overheads on a microscopic level (§7.1), and to benchmark our performance improvement with comparisons to conventional dataset provisioning with synthetic loads (§7.2). To showcase real-world feasibility, usages and performance benefits, we also present our evaluation based on a popular open-source LLM training code base llm.c (§7.3). We will provide the evaluation tools along with the artefacts for reproduction.

All experiments are conducted on a system with dual Intel Xeon Silver 4516Y+ 2.2GHz 24-Core CPUs with Intel TDX enabled, 256 GiB of RAM and a NVIDIA H100 GPU. Both the CVMs and the hypervisor runs Ubuntu 24.0.1 LTS. CVMs are protected under Intel TDX with a stock Linux Kernel 6.8.12 with a patch fixing address resolving and are configured with 4 CPU cores, 8 GiB of RAM and a QEMU `virtio-net-pci` network card. This network card, when uncapped on bandwidth, reflects the maximum possible network speed the system can provide without assigning a physical NIC to the CVM. We chose OpenSSL over Ethernet as the conventional communication channel.

7.1 GPU Travelling Overheads

To demonstrate the overheads imposed by the GPU Travelling mechanism, we present a breakdown of the overheads of each step. We use a synthetic workload that performs data loading from a data holder for various sizes of pre-allocated GPU data buffer. Each time a request is sent, the data holder will fill up the buffer with its dataset. For combination of size and bandwidth, we performed the experiment for 10 times and averaged the time spent for each transmission. The results are presented in Table 1 with a visualised figure in Figure 5.

From the results, we can observe that there were 5 major components that were accounted for most of the overheads: PCIe switching to the data holder, dataset reading, GPU writing, SSL communication and the PCIe switching back to the orchestrator. Since the dataset reading and GPU writing steps are also present in the regular single-node training version, they are considered as an integral part of the training instead of being part of the overheads by the GPU Travelling mechanism. For our GPU Travelling mechanism, the major overheads came from the PCIe MUX switching, which takes approximately 150 ms one way. We can also see that the SSL communication also takes a small fraction of the overheads. The shown results are for an uncapped network reflecting the maximum speed the system can provide and are around 40 ms. We also tested 4 other configurations of bandwidths (4 Mbps, 20 Mbps, 100 Mbps and 500 Mbps) and found out that even for the smallest configuration at 4 Mbps, the overheads were at most 60 ms and therefore would not be significant in the result. This is likely because the keys and contexts were small (about 16 KiB) so the impact from the latency was more significant than the bandwidth.

We see that the overheads of the GPU Travelling mechanism stay almost the same regardless of the size of the dataset buffer and are small enough to be on the same level as the fast PCIe copy. This also provides an insight that the larger the dataset buffer transferred each time, the greater the benefit that GPU Travelling can provide.

7.2 Synthetic Benchmarks

We compared our design with a baseline implementation that transfers all the datasets to the central training node using the same encrypted SSL connection. This represents existing solutions like [6, 55]. To offer insights on how our system can improve on the data transferring, we compared with various configurations of buffer sizes for transmission and bandwidths. The 10-times average of time spent in each transmission are presented in Figure 6. The speed ups (how many times GPU Travelling is faster) are also presented.

From the figure, we can see that for each buffer size, GPU Travelling's overheads remained almost the same because the SSL transmission of our system sends a fixed size context and takes a very small portion of the overheads as discussed in §7.1. On contrast, because the entire datasets needs to be sent over the slow SSL connection, the baseline implementation fluctuates linearly along the bandwidth. This means that a smaller bandwidth will have a significant impact on the baseline one but with only minimal impact on GPU Travelling.

We can also observe that tie-break point between GPU Travelling and the baseline one was around a 4 MiB buffer with over 500 Mbps of bandwidth. If the buffer is larger or the bandwidth is slower, GPU Travelling will be faster than the baseline one. While GPU Travelling imposes a small static overhead, when the bandwidth is very high but the buffer is tiny, there is a chance that SSL speed may overcome that static overhead and overturn the result. However, in practice, as data centre GPUs generally has VRAM counted in GiB level (e.g., H100 has 80 GiB of VRAM), it is not practical to allocate such a small data buffer if GPU Travelling is used nor such

Table 1: Breakdown of the GPU Travelling mechanism. The 'O-' and 'DH-' prefixes mean the step happens in the Orchestrator and the Data Holder respectively. A darker background reflects a heavier overhead.

Buffer Size (MiB)	Overhead (ms)										Total Time (ms)
	O-Clean Up	O-Export Key	DH-PCIe Switching	DH-Import Key	DH-Read Dataset	DH-GPU Write	DH-Export Key	SSL Overheads	O-PCIe Switching	O-Import Key	
4	0.160	0.232	148.029	0.340	2.256	2.976	0.114	32.809	146.647	0.270	333.673
8	0.163	0.229	147.011	0.335	4.369	5.804	0.117	33.321	147.198	0.259	338.642
16	0.198	0.220	146.393	0.338	9.563	12.176	0.116	32.778	146.532	0.242	348.360
32	0.191	0.197	147.867	0.349	19.195	24.336	0.131	48.525	146.794	0.277	387.671
64	0.190	0.165	146.698	0.337	38.464	47.557	0.118	32.466	146.131	0.269	412.204
128	0.224	0.206	145.773	0.371	76.353	92.261	0.122	35.466	146.757	0.279	497.589
256	0.274	0.427	143.073	0.345	147.382	178.816	0.137	35.686	146.134	0.282	652.282
512	0.533	0.206	147.968	0.322	288.922	349.278	0.131	44.112	147.727	0.303	978.969

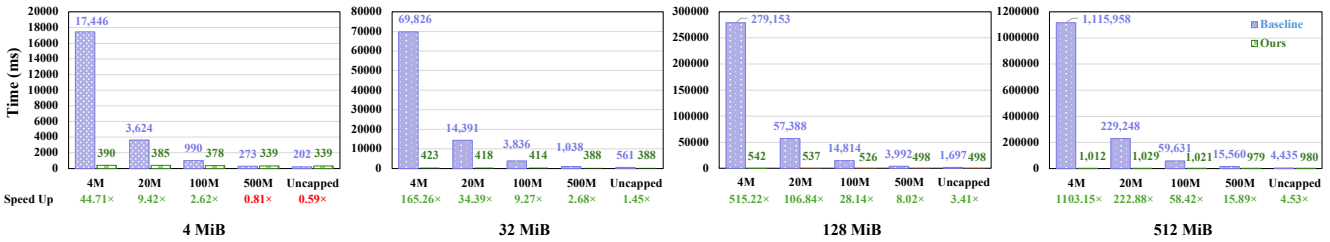

Figure 6: Comparison on synthetic data transmission of different buffer sizes and different bandwidths.

Table 2: The time spent in training and transmission in llm.c.

	Baseline			Ours		
	Training (s)	Tx (s)	Tx Percentage	Training (s)	Tx (s)	Tx Percentage
4M	1230.00	1115.88	47.568%	1230.26	1.01	0.082%
20M	1231.39	230.84	15.787%	1229.39	1.03	0.084%
100M	1231.17	60.36	4.674%	1230.57	1.02	0.083%
500M	1230.73	15.86	1.272%	1229.67	0.98	0.080%
Uncapped	1229.36	7.34	0.594%	1231.46	0.98	0.079%

a high dedicated bandwidth for every data holder. We tested this configuration merely to push our system to the tie-break point.

The results show a clear performance benefit of our system across different bandwidths and buffer sizes with significant improvements particularly for larger buffers and relatively slower bandwidths.

7.3 Application Evaluation with llm.c

To demonstrate real-world feasibility and benefits, we compared our system with llm.c [3], an open-source CUDA-based LLM training project using the GPT-2 model. The model itself has been pre-trained and we are doing the fine-tuning part that uses the same logic as the training to reflect the training process. In this section, we discuss the porting effort and the performance benefits to use GPU Travelling in a real-world large model training scenario.

Porting Effort. In llm.c, just like most training software, the data is loaded using a data loader subsystem. This is the only place we need to modify. Instead of loading from local dataset files, we now load from a data holder *remotely* using our GPU Travelling mechanism as well as an SSL version for comparison. Also in llm.c, the data holder naïvely loads only one batch of data each time. Instead, we now fill the data buffer each time. The rest of the code remains the same.

Performance Benefits. We compared the performance of GPU Travelling using a 512 MiB buffer with different bandwidths. The results are presented in Table 2. We are particularly interested in two criteria: Absolute time and relative percentage.

From the table we can tell that for a single 512 MiB transmission, we can save at least 6.36 seconds even for the uncapped network. This can scale to a considerable amount of time when the training dataset is large since each 512 MiB transmission can save 6.36 seconds and the training involves way more than one epoch.

For relative speed up, we consider how many percentage of time was spent on transmission in the entire training. As we can see, the transmission cost on a baseline implementation reached over 1% even with a relatively high 500 Mbps bandwidth. It quickly grew to 4.67% when the bandwidth was a more common 100 Mbps and kept growing for the two slower bandwidths.

As requiring uncapped bandwidth is typically not the most cost effective deployment configuration, we believe that most of the existing collaborative training mechanisms will need to sacrifice on the transmission overheads in exchange of confidentiality. With GPU Travelling, the cost can be significantly reduced, both on the absolute time consumption and the relative overheads.

7.4 Summary of Results

The takeaway of the evaluation results is that GPU Travelling eliminates the transmission of large dataset via slow connections and imposes a small and fixed overhead that does not grow along the transmission size or the bandwidth. When compared with a baseline implementation, GPU Travelling was significantly faster in most configurations except for some edge cases. The relationship between the speed up, dataset buffer size and bandwidth is: A larger dataset buffer and a slower link contributes to slower transmission

in the baseline implementation, and therefore a more significant speed-up for GPU Travelling. The llm.c integration shows feasibility in real world applications with reasonable porting burden. The performance results showed considerable training time reduction.

Implication of Even Faster Networking. In our evaluation, due to hardware limitations, the uncapped network reaches around 1 Gbps. In specialised deep learning cloud VM instances, inter-VM connection can sometimes go up to a higher bandwidth such as 10 Gbps with physical NICs. We here estimate the implication of a higher bandwidth using our evaluation data. From Figure 6, we found that SSL cryptographic imposed about 30%-40% of overheads for bandwidth over 100 Mbps and the overheads are more significant when the bandwidth is bigger. Therefore, for 10 and 100 Gbps, the overhead should be more than 40%. We take 40% for estimation so the effective bandwidth will be 6 Gbps. For a 1 GiB dataset buffer, 10 Gbps link can take 1.3 s to transmit. The GPU copying will cause approximately 0.7 s and therefore, the total time will be 2 s. For GPU Travelling, the static overhead is about 300 ms and the total time will be 1 s. From the estimation, we can still be 2x faster on a 1 GiB buffer and a 10 Gbps link and save 1 second for every 1 GiB of dataset transmitted.

While our GPU Travelling can still improve the performance considerably even with a faster networking, do note that these high-bandwidth NICs are targeting the training nodes (e.g., the orchestrator in GPU Travelling) for the data exchange in the training process. For data holder CVMs, since it does not have the duty to run computations, it is generally not cost-effective to assign high-bandwidth NICs to them.

8 Related Works

Federated Learning. Federated Learning (FL) [29, 34] is a collaborative training mechanism that enables multiple participants to jointly build a machine learning model while keeping their private data local. Instead of sharing raw data, participants exchange model updates, making FL an attractive solution for mutually distrusted parties or entities handling sensitive information, such as healthcare or financial institutions, where data privacy is critical.

In practice, Federated Stochastic Gradient Descent (FedSGD) [45] and Federated Averaging (FedAvg) [34] are the most widely used aggregation algorithms in FL for training deep neural networks (DNNs). These approaches rely on iterative synchronisation and merging of model updates across training rounds. Recent research has also explored integrating FL with confidential computing [7, 14, 22, 36, 43] to further strengthen security guarantees.

Despite these advantages, FL faces major performance bottlenecks, especially when training large-scale models like LLMs. Each training iteration requires frequent transmission of model updates or gradients over the network, incurring substantial communication overhead. This becomes increasingly prohibitive as model sizes grow. Moreover, when confidential GPUs are used, the need to seal and unseal large models and training data for transfer into the GPU's protected memory imposes additional computational overhead [11, 37], further degrading performance. While compression techniques such as [46, 47, 56] can partially alleviate communication overhead for specific workloads, their effectiveness remains limited in the general case. Generic compression methods typically achieve compression ratios around 0.5, which still leaves a substantial volume of data to be transmitted and is especially problematic for large models like LLMs. Achieving significantly higher compression rates without sacrificing training accuracy or applicability across diverse models and datasets remains a fundamental challenge. As a result, compression alone is insufficient to address the communication bottlenecks in large-scale collaborative training.

Our GPU Travelling approach introduces a fundamentally different collaborative training mechanism. Instead of repeatedly transmitting model updates or training datasets, we deploy a confidential GPU as a secure physical entity that travels between data holders and the orchestrator. The model is loaded once into the GPU's protected memory and remains there throughout training. Each data holder locally feeds their training data directly into the GPU, avoiding transmission over slower networks and eliminating the need for repeated sealing/unsealing operations. This significantly reduces both communication and cryptographic overhead, enabling scalable and confidential collaborative training for large models and datasets, such as those used in LLMs. However, GPU Travelling has a limitation on geographically distributed scenarios as it requires direct PCIe link from CVMs to the GPU. We discuss methods to scale into data-centre level and, with the combination of techniques like FL, to achieve geographical distribution in §9.

GPU Disaggregation. In GPU disaggregation [21], GPUs are decoupled from a single host, grouped into a pool and then connected to a host (or a VM) on demand using hardware/software switches. The disaggregation can be implemented on different layers (e.g., user-space library, driver or PCIe level). When a GPU is assigned to a host or a VM, it typically performs a reset so that the host or the VM can use the GPU as a clean environment. Without the reset, the host or the VM does not have the corresponding driver/library context to communicate with the GPU and the GPU would not function properly [16, 28].

When compared with GPU disaggregation, GPU Travelling uses a similar design based on PCIe link switching. However, our design has to solve the non-trivial problem of keeping the existing contexts and data on the GPU with confidential computing enabled while maintaining the confidentiality of multiple mutual-distrusted data holders.

While there are significant differences, GPU disaggregation can work with our GPU Travelling mechanism to receive benefits from both worlds. In particular, disaggregation methods based on PCIe links can have their PCIe switching mechanism used as the PCIe MUX in GPU Travelling. With data holders CVMs properly setup, workloads on disaggregated GPUs can also have the assigned GPU to travel to these data holder CVMs for dataset collection. The disaggregation also implies that GPUs are no longer restricted within one physical host. This can enable even more possibilities to be discussed in §9 when combined with GPU Travelling.

9 Future Works and Outlooks

PCIe TDISP Integration. Future revision of PCIe (PCIe 6.0) has a feature called TEE Device Interface Security Protocol (TDISP) [42] that offers encryption on the PCIe protocol level instead of using the bounce buffer. This protocol will allow a unified and encrypted PCIe connection between a CVM and an enlightened device. We

believe that the same idea of GPU Travelling can also be applied to this feature. Changes may need to be made on the key migration part of the context travelling. This time, the PCIe key needs to be exported. However, due to lack of commercially available hardware, we leave this as a future work.

Workload Scaling. Our GPU Travelling mechanism is designed to focus on the lower level data transmission layer and therefore can be combined with many of the existing GPU workload scaling techniques. For example, in model parallelism [5], each GPU trains a portion of the model through the entire dataset. This means that each GPU can travel to every data holder on its own to collect data and train. Another example like in data parallelism [27], each GPU trains the entire model on a small portion of the dataset. In this case, each GPU can travel to only those data holders that has the data it needs. However, since NVIDIA's confidential computing currently does not support passing multiple confidential GPUs into a single CVM, we leave this as a future work.

Hardware PCIe Switching. In our current implementation, the GPU resides on a single host server and is physically attached to that server. This limits the GPU's range of travel to be within that server. However, hardware PCIe switches (e.g., PCIe fabrics) are available [24] and may be employed to connect GPUs to multiple servers on demand as in the GPU disaggregation discussed above, we believe that the same hardware PCIe switches can also be used to allow a GPU to travel to data holder CVMs on another physical server in the data centre.

Distribution Beyond Data-Centre Level. GPU Travelling requires the GPU used to be reachable by PCIe links to data holder CVMs and the orchestrator. With PCIe fabrics discussed above, GPU Travelling can be applied for distribution up to a data-centre-level. However, when geographical distribution is needed, as a low-level mechanism, GPU Travelling can also be easily combined with existing geographically distributed techniques like FL in a two-layer fashion. For example, data holders in a single data centre can form a local cohort in which they could employ GPU Travelling. These geographically-distributed cohorts can then still use FL by integrating the logics into each cohort's orchestrator. By doing this, each member of the local cohort no longer needs to send large model gradients across the network on its own when compared to vanilla FL techniques. In this way, one can push the collaborative training beyond data-centre-level while still benefit from our GPU Travelling technique.

Dataset-as-a-Service. In confidential CVMs, live migration has been researched and available to allow a CVM to migrate from one server to another [23, 51]. We envision that a data holder CVM can be provisioned using the live migration on demand to a specific training server where the the orchestrator on that server can then use the datasets in the specific data holder CVM with our GPU Travelling mechanism. We believe this can enable a new service scheme as dataset-as-a-service in which datasets can be securely used for training without the worry of leaking.

Changes in Proprietary Components. We set off the project with the restriction that no proprietary components may be changed to achieve GPU Travelling so that a practical system can be offered straight out of the box with existing hardware and software. We now discuss new possibilities can be achieved by changing certain proprietary components.

- **Limit GPU Memory Access.** This can be done by changing the GPU's firmware so that a CE can be limited to access only certain region of the GPU memory. When this is possible, the orchestrator can pick a CE for the data holder and limit the CE's access to only the dataset buffer. When GPU travels, only the key of this CE will be provisioned so that the data holder cannot read or write the model or other memory. This can eliminate certain side channel attacks like model inferencing.
- **CUDA Context Migration.** By modifying the CUDA runtime to export CUDA contexts together with GPU contexts, we technically can allow the GPU to travel permanently and completely to a new CVM. This can be achieved either by migrating the entire context memory with the approach in VMST [17] or by transferring only the necessary portions. If this is possible, whenever a CVM is no longer capable of running the training, we can keep the training on the GPU and travel to a new CVM for uninterrupted service. For example, a CVM may need maintenance and this may happen in the middle of a training; Or, a CVM's configuration may turn out to be unsuitable for the specific training workload in the middle of a training and a new and better-configured CVM might be desired.
- **Orchestrator Fail-Safe.** CUDA context migration can also benefit the availability in the GPU Travelling directly. In our design, there is only a single orchestrator and its failure may halt the entire training process. To achieve fail-safe, we need redundancy in the orchestrator which requires duplicated orchestrator nodes. Since the orchestrator CVM requires full CUDA context, similar to the CUDA context migration, we can modify the CUDA runtime to export the CUDA context to the new orchestrator node. There can be periodic training context sync-ups between these redundant orchestrator nodes so that if one of them fails, a redundant orchestrator can immediately take over the training to achieve fail-safe.

10 Conclusion

We have presented GPU Travelling, a novel mechanism that enables efficient confidential collaborative training with TEE-enabled GPUs by allowing GPUs to securely travel to data holders to collect dataset, and therefore eliminates the encrypted large dataset/model transmission over slower connections. Our implementation and integration with llm.c demonstrated the feasibility to apply GPU Travelling in a real world setting. The evaluation results showed that GPU Travelling outperformed conventional methods for common data buffer sizes and networking bandwidths, and can significantly benefit in both relative performance and absolute time saving for modern collaborative training workloads like LLM.

Acknowledgement

We would like to thank the anonymous reviewers for their feedback and suggestions. The authors from The Ohio State University were partially supported by NSF awards 2112471, 2207202, and 2348754. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Amazon. 2023. Amazon EC2 now supports AMD SEV-SNP. https://aws.amazon.com/about-aws/whats-new/2023/04/amazon-ec2-amd-sev-snp/.

[2] AMD. 2020. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. *White paper* (2020).

[3] Andrej Karpathy. [n. d.]. karpathy/llm.c: LLM training in simple, raw C/CUDA. https://github.com/karpathy/llm.c.

[4] Azure. 2023. Preview: Introducing DCesv5 and ECesv5-series Confidential VMs with Intel TDX. https://azure.microsoft.com/en-us/updates/confidential-vms-with-intel-tdx-dcesv5-ecesv5/.

[5] Felix Brakel, Uraz Odyurt, and Ana-Lucia Varbanescu. 2024. Model Parallelism on Distributed Infrastructure: A Literature Review from Theory to LLM Case-Studies. arXiv:2403.03699 [cs.DC] https://arxiv.org/abs/2403.03699

[6] Dong Chen, Alice Dethise, Istemi Ekin Akkus, Ivica Rimac, Klaus Satzke, Antti Koskela, Marco Canini, Wei Wang, and Ruichuan Chen. 2024. Protecting Confidentiality, Privacy and Integrity in Collaborative Learning. arXiv:2412.08534 [cs.DC] https://arxiv.org/abs/2412.08534

[7] Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, KR Jayaram, Enriquillo Valdez, and Ashish Verma. 2024. Deta: Minimizing data leaks in federated learning via decentralized and trustworthy aggregation. In *Proceedings of the nineteenth european conference on computer systems*. 219–235.

[8] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. *Comput. Surveys* 56, 9 (2024), 1–33.

[9] Christopher A. Choquette-Choo, Natalie Dullerud, Adam Dziedzic, Yunxiang Zhang, Somesh Jha, Nicolas Papernot, and Xiao Wang. 2021. CaPC Learning: Confidential and Private Collaborative Learning. arXiv:2102.05188 [cs.LG] https://arxiv.org/abs/2102.05188

[10] Katharine Daly, Hubert Eichner, Peter Kairouz, H. Brendan McMahan, Daniel Ramage, and Zheng Xu. 2025. Federated Learning in Practice: Reflections and Projections. arXiv:2410.08892 [cs.LG] https://arxiv.org/abs/2410.08892

[11] Gobikrishna Dhanuskodi, Sudeshna Guha, Vidhya Krishnan, Aruna Manjunatha, Rob Nertney, Michael O'Connor, and Phil Rogers. 2023. Creating the first confidential GPUs. *Commun. ACM* 67, 1 (2023), 60–67.

[12] DMTF. [n. d.]. SPDM | DMTF. https://www.dmtf.org/standards/spdm.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography* (New York, NY) *(TCC'06)*. Springer-Verlag, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

[14] Hubert Eichner, Daniel Ramage, Kallista Bonawitz, Dzmitry Huba, Tiziano Santoro, Brett McLarnon, Timon Van Overveldt, Nova Fallen, Peter Kairouz, Albert Cheu, et al. 2024. Confidential federated computations. *arXiv preprint arXiv:2404.10764* (2024).

[15] Hugging Face. 2020. OpenAI GPT2. https://huggingface.co/docs/transformers/model_doc/gpt2.

[16] Henrique Fingler, Zhiting Zhu, Esther Yoon, Zhipeng Jia, Emmett Witchel, and Christopher J. Rossbach. 2022. DGSF: Disaggregated GPUs for Serverless Functions. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 739–750. https://doi.org/10.1109/IPDPS53621.2022.00077

[17] Yangchun Fu and Zhiqiang Lin. 2012. Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In *2012 IEEE symposium on security and privacy*. IEEE, 586–600.

[18] Daniel J Gervais, Noam Shemtov, HARALAMBOS MARMANIS, and CATHERINE ZALLER ROWLAND. 2024. The Heart of the Matter: Copyright, AI Training, and LLMs. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4963711. (2024).

[19] Google. 2020. Introducing Google Cloud Confidential Computing with Confidential VMs. https://cloud.google.com/blog/products/identity-security/introducing-google-cloud-confidential-\computing-with-confidential-vms.

[20] Zhongshu Gu, Enriquillo Valdez, Salman Ahmed, Julian James Stephen, Michael Le, Hani Jamjoom, Shixuan Zhao, and Zhiqiang Lin. 2025. NVIDIA GPU Confidential Computing Demystified. arXiv:2507.02770 [cs.CR] https://arxiv.org/abs/2507.02770

[21] Anubhav Guleria, J. Lakshmi, and Chakri Padala. 2019. EMF: Disaggregated GPUs in Datacenters for Efficiency, Modularity and Flexibility. In *2019 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 1–8. https://doi.org/10.1109/CCEM48484.2019.000-5

[22] Jinnan Guo, Peter Pietzuch, Andrew Paverd, and Kapil Vaswani. 2024. Trustworthy AI using Confidential Federated Learning: Federated learning and confidential computing are not competing technologies. *Queue* 22, 2 (2024), 87–107.

[23] Pankaj Gupta and Tom Lendacky. 2023. SEV-SNP Live Migration and VMM/KVM API Implications. https://lpc.events/event/17/contributions/1532/attachments/1369/2974/06%20LPC-SNP-Live-Migration.pdf

[24] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. 2024. Understanding Routable PCIe Performance for Composable Infrastructures. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 297–312. https://www.usenix.org/conference/nsdi24/presentation/hou

[25] IBM. 2025. Confidential computing solutions. https://www.ibm.com/confidential-computing.

[26] Intel. 2020. Intel Trust Domain Extensions Whitepaper. https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf.

[27] Jinda Jia, Cong Xie, Hanlin Lu, Daoce Wang, Hao Feng, Chengming Zhang, Baixi Sun, Haibin Lin, Zhi Zhang, Xin Liu, and Dingwen Tao. 2024. SDP4Bit: Toward 4-bit Communication Quantization in Sharded Data Parallelism for LLM Training. arXiv:2410.15526 [cs.LG] https://arxiv.org/abs/2410.15526

[28] Xin Jin, Zhihao Bai, Zhen Zhang, Yibo Zhu, Yinmin Zhong, and Xuanzhe Liu. 2024. DistMind: Efficient Resource Disaggregation for Deep Learning Workloads. *IEEE/ACM Trans. Netw.* 32, 3 (Jan. 2024), 2422–2437. https://doi.org/10.1109/TNET.2024.3355010

[29] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and trends® in machine learning* 14, 1–2 (2021), 1–210.

[30] David Kaplan. 2017. Protecting VM register state with SEV-ES. *White paper* (2017).

[31] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[32] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492 [cs.LG] https://arxiv.org/abs/1610.05492

[33] Hiroki Masuda, Kentaro Kita, Yuki Koizumi, Junji Takemasa, and Toru Hasegawa. 2021. Model Fragmentation, Shuffle and Aggregation to Mitigate Model Inversion in Federated Learning. In *2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 1–6. https://doi.org/10.1109/LANMAN52105.2021.9478813

[34] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[35] Timo Minssen, Sara Gerke, Mateo Aboy, Nicholson Price, and Glenn Cohen. 2020. Regulatory responses to medical machine learning. *Journal of Law and the Biosciences* 7, 1 (2020), lsaa002.

[36] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2022. Ppfl: Enhancing privacy in federated learning with confidential computing. *GetMobile: Mobile Computing and Communications* 25, 4 (2022), 35–38.

[37] 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)*. IEEE, 164–175.

[38] NVIDIA. [n. d.]. NVIDIA/open-gpu-kernel-modules: NVIDIA Linux open GPU kernel module source. https://github.com/NVIDIA/open-gpu-kernel-modules.

[39] NVIDIA. 2024. Confidential Computing | NVIDIA. https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/.

[40] NVIDIA. 2024. Confidential Computing Deployment Guide - (Intel TDX & KVM). https://docs.nvidia.com/cc-deployment-guide-tdx.pdf.

[41] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. 2018. SoK: Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 399–414. https://doi.org/10.1109/EuroSP.2018.00035

[42] PCI-SIG. [n. d.]. IDE and TDISP: An Overview of PCIe® Technology Security Features | PCI-SIG. https://pcisig.com/blog/ide-and-tdisp-overview-pcieÂő-technology-security-features.

[43] Do Le Quoc and Christof Fetzer. 2021. Secfl: Confidential federated learning using tees. *arXiv preprint arXiv:2110.00981* (2021).

[44] Wolfram Ravenwolf. 2025. LLM Comparison/Test: DeepSeek-V3, QVQ-72B-Preview, Falcon3 10B, Llama 3.3 70B, Nemotron 70B in my updated MMLU-Pro CS benchmark. https://huggingface.co/blog/wolfram/llm-comparison-test-2025-01-02.

[45] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1310–1321.

[46] Jiajun Song, Jiajun Luo, Rongwei Lu, Shuzhao Xie, Bin Chen, and Zhi Wang. 2024. A Joint Approach to Local Updating and Gradient Compression for Efficient Asynchronous Federated Learning. arXiv:2407.05125 [cs.DC] https://arxiv.org/abs/2407.05125

[47] Haijian Sun, Xiang Ma, and Rose Qingyang Hu. 2020. Adaptive Federated Learning With Gradient Compression in Uplink NOMA. arXiv:2003.01344 [cs.NI] https://arxiv.org/abs/2003.01344

[48] Shiyu Sun, Shu Wang, Xinda Wang, Yunlong Xing, Elisa Zhang, and Kun Sun. 2023. Exploring Security Commits in Python . In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, Los

Alamitos, CA, USA, 171–181. https://doi.org/10.1109/ICSME58846.2023.00027
[49] Andrew S. Tanenbaum. 1989. Computer Networks. (1989), 57.
[50] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A Hybrid Approach to Privacy-Preserving Federated Learning. arXiv:1812.03224 [cs.LG] https://arxiv.org/abs/1812.03224
[51] Wei Wang. 2021. TDX Live Migration. https://lpc.events/event/11/contributions/960/attachments/839/1586/TDX%20Live%20Migration_Wei%20Wang.pdf
[52] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. 2020. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3454–3469. https://doi.org/10.1109/TIFS.2020.2988575
[53] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. 2025. On protecting the data privacy of Large Language Models (LLMs) and LLM agents: A literature review. *High-Confidence Computing* (2025), 100300. https://doi.org/10.1016/j.hcc.2025.100300
[54] 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 *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 1210–1228. https://doi.org/10.1145/3689031.3717464
[55] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. 2021. Citadel: Protecting Data Privacy and Model Confidentiality for Collaborative Learning with SGX. arXiv:2105.01281 [cs.CR] https://arxiv.org/abs/2105.01281
[56] Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. 2023. FedPETuning: When Federated Learning Meets the Parameter-Efficient Tuning Methods of Pre-trained Language Models. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 9963–9977. https://doi.org/10.18653/v1/2023.findings-acl.632
[57] Shixuan Zhao, Mengyuan Li, Yinqian Zhangyz, and Zhiqiang Lin. 2022. vSGX: Virtualizing SGX Enclaves on AMD SEV. In *2022 IEEE Symposium on Security and Privacy (SP)*. 321–336. https://doi.org/10.1109/SP46214.2022.9833694
[58] Shixuan Zhao, Pinshen Xu, Guoxing Chen, Mengya Zhang, Yinqian Zhang, and Zhiqiang Lin. 2023. Reusable Enclaves for Confidential Serverless Computing. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 4015–4032. https://www.usenix.org/conference/usenixsecurity23/presentation/zhao-shixuan
[59] Mingwei Zheng, Chengpeng Wang, Xuwei Liu, Jinyao Guo, Shiwei Feng, and Xiangyu Zhang. 2025. An LLM Agent for Functional Bug Detection in Network Protocols. arXiv:2506.00714 [cs.SE] https://arxiv.org/abs/2506.00714