PipeLLM: Fast and Confidential Large Language Model Services with Speculative Pipelined Encryption

Yifan Tan
Institute of Parallel and Distributed Systems
SEIEE, Shanghai Jiao Tong University
Shanghai, China
tanyifan@sjtu.edu.cn

Cheng Tan
Northeastern University
Boston, USA
c.tan@northeastern.edu

Zeyu Mi
Institute of Parallel and Distributed Systems
SEIEE, Shanghai Jiao Tong University
Shanghai, China
yzmizeyu@sjtu.edu.cn

Haibo Chen
Institute of Parallel and Distributed Systems
SEIEE, Shanghai Jiao Tong University
Shanghai, China
haibochen@sjtu.edu.cn

Abstract

Confidential computing on GPUs, like NVIDIA H100, mitigates the security risks of outsourced Large Language Models (LLMs) by implementing strong isolation and data encryption. Nonetheless, this encryption incurs a significant performance overhead, reaching up to 52.8% and 88.2% throughput drop when serving OPT-30B and OPT-66B, respectively.

To address this challenge, we introduce *PipeLLM*, a user-transparent runtime system. PipeLLM removes the overhead by overlapping the encryption and GPU computation through pipelining—an idea inspired by the CPU instruction pipelining—thereby effectively concealing the latency increase caused by encryption. The primary technical challenge is that, unlike CPUs, the encryption module lacks prior knowledge of the specific data needing encryption until it is requested by the GPUs. To this end, we propose *speculative pipelined encryption* to predict the data requiring encryption by analyzing the serving patterns of LLMs. Further, we have developed an efficient, low-cost pipeline relinquishing approach for instances of incorrect predictions. Our experiments show that compared with vanilla systems without confidential computing (e.g., vLLM, PEFT, and FlexGen), PipeLLM incurs modest overhead (<19.6% in throughput) across various LLM sizes, from 13B to 175B. PipeLLM's source code is available at https://github.com/SJTU-IPADS/PipeLLM.

ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-0698-1/25/03
https://doi.org/10.1145/3669940.3707224

CCS Concepts: • **Security and privacy → Systems security**; • **Computing methodologies → Machine learning**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

Keywords: Nvidia Confidential Computing; Large Language Model; Confidential Virtual Machine

ACM Reference Format:
Yifan Tan, Cheng Tan, Zeyu Mi, and Haibo Chen. 2025. PipeLLM: Fast and Confidential Large Language Model Services with Speculative Pipelined Encryption. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3669940.3707224

1 Introduction

Large Language Models (LLMs) are increasingly used across various applications [33, 53]. With the growth of open-source LLMs [23, 48, 52], companies are integrating and fine-tuning these models into their business operations.

Due to LLM's reliance on high-end GPUs, many businesses opt for outsourced services, such as cloud, attracted by their high availability and flexible pay-as-you-go models. However, these cloud infrastructures, often complex in nature, encompass a large Trusted Computing Base (TCB), which may contain vulnerabilities, both publicly reported and zero-day [8, 28]. This poses security risks for LLMs, which are usually fine-tuned with proprietary data, and user prompts that contain sensitive business information. Thus, any data breach could expose critical business secrets.

To mitigate these security threats, people introduce confidential computing. Confidential computing is designed to safeguard tenants' code and data against untrusted privileged software and rogue employees of cloud providers. The confidential virtual machine (CVM), supported by technologies such as Intel TDX [18], AMD SEV [2], and ARM CCA [3],

serves as a prime example of this. Any software external to a CVM is unable to access the code and data within it.

Regarding machine learning workloads, people develop GPU enclaves to enhance security measures within GPUs [36, 50]. A notable implementation of this is the NVIDIA H100 GPU [36], which supports confidential computing inside the GPU to protect sensitive data and models from unauthorized access. Moreover, the data communication between the CVM and the GPU enclave is encrypted, further reinforcing the security of I/O operations.

Although GPU confidential computing effectively enhances security for traditional small-scale AI models, it significantly undermines the performance of LLMs in throughput and latency. Our comprehensive experiments on NVIDIA H100 GPUs reveal that the GPU enclave can incur up to a 52.8% latency overhead on serving OPT-30B, a 36.2% throughput drop on fine-tuning OPT-30B, and an 88.2% throughput drop on serving OPT-66B (§3). This overhead is largely due to a combination of memory swapping plus encryption. The swapping happens because LLMs consume a huge amount of GPU memory. For example, the OPT-66B model needs approximately 132GB of memory to store all its parameters, surpassing the 80GB memory of H100 GPUs. Moreover, runtime states such as the Key-Value cache (KV cache) [38] during LLM inferences and activation during LLM training also consume significant GPU memory.

Owing to the limited GPU memory, a GPU enclave has to dynamically swap out inactive parameters and/or runtime states to the main memory. This process requires encrypting the data transferred out of the GPU enclave. Correspondingly, the CPU cores must decrypt data received from the GPU, and re-encrypt it before sending it back to the GPU (§2.2). However, the encryption and decryption pose a severe bottleneck due to the limited computational capability. This bottleneck significantly harms the overall performance, particularly in the context of LLMs.

This paper introduces PipeLLM, a system designed to eliminate the performance overhead associated with GPU confidential computing for LLMs. Importantly, PipeLLM achieves this without requiring any changes to the existing LLM systems or the hardware, while still upholding the same level of security. The underlying principle of PipeLLM is straightforward yet effective: it decouples encryption tasks from the critical path of the memory swapping mechanism, by leveraging *speculative pipelined encryption* (§4.3), a technique we proposed. Drawing inspiration from the concept of speculative execution in CPUs, PipeLLM anticipates which data blocks will be required by the GPU and pre-encrypts them. By doing so, PipeLLM significantly reduces the overhead of the GPU confidential computing by integrating predictions, encryptions, and data transfers into a pipeline.

However, akin to CPU pipelining, an incorrect prediction could not only waste an individual pre-encrypted data but also invalidate the entire pipeline of subsequent pre-encrypted data. This is a consequence of the encryption scheme used by GPU enclaves, designed to prevent replay attacks [34]. In the H100's confidential computing, data is encrypted using a private key in conjunction with a unique integer known as the *Initialization Vector (IV)*. The IV is synchronized between the CPU and GPU, and increments by one with each encryption. Consequently, if an incorrect piece of data is encrypted in a pipeline, all subsequent IVs in the pipeline could become invalid, requiring re-encryption of the subsequent data with the correct IVs. We will elaborate on the encryption mechanism and IVs in §5.3.

To address the challenge, we observe that LLM systems are highly predictable in swapping, allowing PipeLLM to accurately determine the sequence of data being swapped using heuristics. For instance, FlexGen [42] and PEFT [31] compute the LLM through a layer-by-layer process, enabling PipeLLM to efficiently swap in layer parameters in their respective order. Similarly, vLLM [25] uses simple swapping policies like FIFO (First-In, First-Out) and LIFO (Last-In, First-Out). PipeLLM can recognize these swapping policies and use them to predict the future swapping sequence.

Moreover, PipeLLM incorporates several techniques to accelerate the pipeline and mitigate the cost of prediction errors. First, PipeLLM develops an efficient validation scheme to verify the correctness of a pre-defined ciphertext (§5.2). Second, PipeLLM introduces *request re-ordering* and *NOP padding* to handle IV mismatches without relinquishing the entire pipeline (§5.3). Finally, PipeLLM provides asynchronous decryption to accelerate data transfer (§5.4).

We implement PipeLLM with approximately 1K lines of code in C++. We conducted our performance experiments on an Intel server equipped with an NVIDIA H100-SXM GPU. The evaluation results show that PipeLLM significantly reduces the overhead associated with GPU confidential computing for LLM serving and fine-tuning, cutting it from as much as 88.2% to <19.6% in throughput, across various LLM sizes, ranging from 13 billion to 175 billion parameters.

In summary, this paper makes the following contributions:

- We conduct a comprehensive performance analysis of NVIDIA Confidential Computing on an H100 GPU enclave with LLM workloads.
- We propose speculative pipelined encryption, an approach to greatly reduce the swapping overhead of confidential computing. It works well for LLM serving and fine-tuning.
- We have built a system PipeLLM and evaluated its performance on multiple state-of-the-art LLM systems.

2 Background

2.1 Large Language Models (LLMs)

LLMs are language models that take a sequence of text as input and produce a corresponding sequence of text as output. The Transformer architecture [49] is the predominant

framework in LLM design, used by well-known models like GPT [7], OPT [52], and LLaMA [48]. An LLM typically consists of an input embedding layer, numerous transformer layers, and an output embedding layer.

The inference process of LLMs begins with a prompt (a series of tokens). Outputs are generated iteratively, with each iteration processing the sequence through all layers to produce the next token. This iterative process, known as auto-regressive generation, continues until an end-of-sequence token is produced or the output length reaches a threshold.

Training an LLM from scratch is very expensive [43]. Instead, fine-tuning [16] a pre-trained LLM is a more viable option for many because fine-tuning requires far fewer resources and less time. This paper aims at serving and fine-tuning LLMs with fast confidential computing.

KV cache in LLM. In LLM inference, the attention mechanism captures the relationship between the current input token and preceding tokens by calculating their context as key and value vectors. Storing these vectors in a *KV cache* [38] allows LLMs to save repetitive calculations, significantly boosting inference efficiency. However, this efficiency comes at the cost of increased GPU memory consumption.

Memory pressure in LLMs. The size of LLMs has been rapidly increasing, outpacing the growth in GPU memory capacity. This disparity has made limited GPU memory a significant bottleneck for serving and fine-tuning LLMs. The memory pressure associated with LLMs originates from three primary sources. First, both the inference and fine-tuning of LLMs require the entire set of parameters for computation, and the model weights may exceed the GPU's available memory capacity. Second, for serving LLMs, KV cache consumes a lot of memory, for example, vLLM allocates about 30% of GPU memory for KV cache [25]. Third, activation and optimizer states during fine-tuning are memory-intensive. The size of the activation state is proportional to the batch size, input length, and model hidden size, while the optimizer state size depends on the number of trainable parameters.

GPU memory swapping. To mitigate the LLM memory pressure, previous studies have introduced various GPU swapping techniques. FlexGen [42] only loads the parameters of the active layer onto the GPU, while maintaining the parameters of other layers in CPU memory. vLLM [25] addresses memory shortages by pausing some running requests, swapping their KV cache to main memory, and later loading back the vectors to resume processing these requests. Deepspeed [39] facilitates optimizer offloading and model offloading to free up memory, enabling larger batch sizes and higher throughput in pre-training and fine-tuning LLMs.

2.2 Confidential Computing (CC)

Cloud computing is a fundamental component of today's digital world. Confidential computing has emerged as a solution

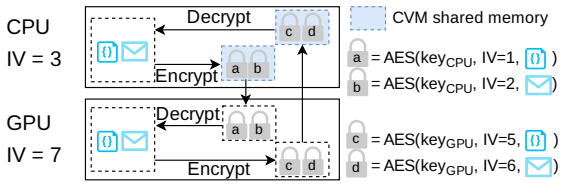


Figure 1: Workflow of encrypted data transfer in NVIDIA CC. The messages labeled "a" and "b" represent two consecutive ciphertexts transferred from the CPU to the GPU, while "c" and "d" denote ciphertexts moved from the GPU back to the CPU. After these transfers, the CPU and GPU have IVs of 3 and 7, respectively.

for securely outsourcing computation to the cloud without compromising security and privacy. Cloud providers like Microsoft Azure [4, 5] and Google Cloud [10] have introduced *confidential virtual machines (CVMs)* as a key measure for confidential computing. CVMs ensure robust isolation from untrusted hypervisors, enhancing security and privacy with minimal changes to existing cloud infrastructure.

Confidential VMs (CVMs). A CVM, such as Intel TDX [18] and AMD SEV-SNP [2], isolates the user's OS (the guest) from the hypervisor (the host) and secures the user's code and data without modifying applications. Additionally, CVMs encrypt their memory to guarantee privacy. CVMs generally incur minimal performance overhead, averaging around 4% [24]. I/O operations however may introduce significant performance overhead due to data copying and encryption [27].

Confidential Computing (CC) on GPUs. Beyond CPU-based CVMs, confidential computing on GPUs secures GPU computations such as LLM serving and training. Soter [40], designed for edge computing, uses CPU-side confidential computing to eliminate the trust on GPU hardware. It employs parameter morphing to offload certain layers to GPUs, ensuring model weights secrecy. However, Soter imposes significant overhead (2× to 15×) on LLM inference tasks, as non-associative layers like embedding must be processed on CPUs. Alternatively, Honeycomb [29] introduces a trusted software layer and static validation to block insecure GPU commands, but this adds substantial performance overhead due to extra runtime checks for indirect memory access, heavily used in systems like vLLM [25].

Unlike software-based solutions, NVIDIA Confidential Computing relies on hardware: NVIDIA H100 GPU is the first commercial implementation with confidential computing capability [15]. Working with CVMs, H100 could build a GPU enclave, allowing users to have exclusive control over the GPU and rejecting any access from the host, such as read/write GPU memory and modify the control flow. Hardware GPU confidential computing has low performance overhead and is backward-compatible with existing applications. This paper focuses on studying hardware GPU confidential computing.

A closer look at NVIDIA CC. Although CVMs encrypt their memory, this encryption is separate from that used

by NVIDIA CC. NVIDIA CC ensures the confidentiality and integrity of communication between a CVM and a GPU via AES-GCM encryption [15]. A critical component of AES-GCM is the *Initialization Vector* (IV), a unique, non-repeating number (a nonce) required for each encryption session. Both the CVM and GPU maintain and increment their own IVs for their respective encryption sessions.

Figure 1 illustrates the workflow of data transfers of the NVIDIA CC. Consider the process of transferring memory from the CPU to the GPU, which is similar to the reverse operation from GPU to CPU. All CPU-side application data resides in the CVM's private encrypted memory. To transfer data from the CPU to the GPU, the application invokes CUDA APIs (e.g., `cudaMemcpyAsync`). The CUDA library detects that it is operating in CC mode and encrypts the data using the AES-GCM key and IV before sending it to the GPU. The encrypted data is then copied to CVM shared memory, allowing the GPU to perform DMA.

During data transfer, the IV is not explicitly transmitted with each operation. Instead, both sides only synchronize their initial IVs during system initialization. When the CPU sends data to the GPU, both sides increment their locally-maintained CPU IV; similarly, when the GPU sends data to the CPU, both sides increment their GPU IV. This design eliminates the overhead of IV synchronization for each data transfer while maintaining security. Once the GPU receives the ciphertext from the CPU, the GPU's copy engine—a piece of hardware—decrypts the data and transfers the plaintext to GPU memory. The entire workflow is handled by the CUDA library and the GPU, requiring no modifications to the application code.

3 Analysis of LLMs on GPU Enclave

NVIDIA reports that compared with confidential computing disabled (CC-disabled), confidential computing enabled GPUs have significant performance overhead on IO-intensive workloads like ResNet50 training [15]. This is not surprising, as all I/Os need to be encrypted, and the encryption is on the critical path. On the contrary, LLM serving is supposed to be compute-intensive, and hence has light overhead. As shown by NVIDIA, there is negligible overhead for serving BERT[14] (a language model) on CC-enabled GPUs [15].

However, low-overhead serving or training is no longer true in LLM era. Serving and fine-tuning LLMs is indeed compute-intensive, but it is even more of a *memory-intensive* workload. We have observed that CC-enabled GPUs with 80GB GPU memory can have up to an 88.2% serving throughput drop on OPT-66B model (Figure 3a), a 36.2% fine-tuning throughput drop on OPT-30B model (Figure 3c), and a 52.8% serving capability drop on OPT-30B model (Figure 3b). This is due to the IO caused by memory swapping: GPUs need to swap in and out memory for large LLM inference and fine-tuning. We observe that there are two major reasons for swapping: (1) model offloading and (2) KV cache swapping. In this section, we dive into the details of the two reasons with three state-of-the-art systems, FlexGen [42], PEFT [31] and vLLM [25].

Microbenchmark. Before evaluating the performance of application workloads on NVIDIA Confidential Computing, we conduct a microbenchmark to clarify the performance overhead of I/O operations. This microbenchmark assesses latency and throughput across different I/O sizes. We demonstrate memory copying from host to device, noting that the performance of the reverse operation is similar.

		I/O size			
		32B	128KB	1MB	32MB
Latency (μs)	CC-disabled	1.43	1.17	1.19	1.43
	CC-enabled	14.93	22.809	162.5	5252.1
Throughput (GB/s)	CC-disabled	–	27.16	48.2	55.31
	CC-enabled	–	3.32	5.82	5.83

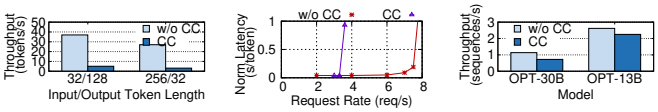
Figure 2: Host To Device Memory Copy of Different Data Size

Figure 2 shows the result. In the table, "Latency" measures the time from the invocation to the return of the host-to-device CUDA API, while "Throughput" indicates the average throughput over 10K transfers. We have omitted the throughputs for small data transfers ("32B") because the control-plane overhead is dominant in this case, and hence both throughputs are tiny. Notably, the throughput of a CC-enabled GPU is approximately an order of magnitude lower than that of CC-disabled, limited by the CPU's encryption capability. Furthermore, the API latency for large packets remains relatively constant in a CC-disabled GPU but increases proportionally in a CC-enabled GPU. This suggests that encryption and decryption processes are coupled with the API call.

Next, we examine three systems—FlexGen [42], vLLM [25], and PEFT [31]—to further illustrate the bottleneck.

Case study 1: serving large LLMs with model offloading. We first experiment with FlexGen and measure its throughputs. FlexGen is the state-of-the-art LLM inference system that supports serving large models whose sizes exceed the GPU memory capacity. FlexGen prefers throughput over latency. We run an OPT-66B model, which requires 132GB of memory, exceeding the 80GB GPU memory capacity of the H100. Thus, FlexGen needs to partially offload the model to CPU memory. We use two configurations: input length 32 with output length 128, and input length 256 with output length 32. To study model offloading, we guarantee all KV cache is stored in GPU memory to avoid KV cache swapping. Other configuration parameters are tuned to have the largest batch size for high throughput on the CC-disabled system.

Figure 3a shows the result. FlexGen's throughput has dropped by up to 88.2% when confidential computing is enabled. Note that CC-enabled and CC-disabled GPUs have



(a) FlexGen **(b)** vLLM **(c)** PEFT

Figure 3: Confidential computing overhead study. Figure 3a and 3c shows the throughput of FlexGen and PEFT under CC-disabled and CC-enabled environments. Figure 3b shows the normalized latency of vLLM under the same configurations.

the same IO; the difference is that the CC-disabled GPU does not need to encrypt the data. Additionally, we measure that FlexGen uses about 64GB/s PCIe bandwidth when CC is disabled; whereas with CC enabled, it only uses approximately 6 GB/s bandwidth, which is bottlenecked by encryption.

The above experiments show that serving large LLMs that exceed available GPU memory imposes significant overhead when confidential computing is enabled. The primary reason is the necessity for model offloading, which in turn introduces significant IOs. This, in turn, results in heavy encryptions, ultimately leading to high overhead.

Case study 2: KV cache swapping. Beyond large LLMs with model sizes exceeding GPU memory, we observe that even when LLM models fit GPU memory, swapping still exists. This is due to intermediate data management: when LLMs are deployed (like serving as chatbots) and interact with the external world, the dynamic inputs, outputs, and their intermediate data trigger memory swapping, in particular KV cache (§2.1).

To investigate the mid-sized LLM serving, we study vLLM, the state-of-art inference system targeting low-latency LLM serving. Unlike FlexGen, vLLM does not offload model weights; instead, it swaps intermediate data to handle the memory pressure caused by the dynamics of serving multiple requests. vLLM targets low latency. We experiment vLLM with an OPT-30B model (60GB). We configure vLLM with parallel sampling, a common decoding policy, and ask vLLM to generate 6 output sequences for each input.

Figure 3b shows the latency changes when increasing request rates for CC-enabled and CC-disabled GPUs. When the request rate is low, they have similar performance because there is no memory pressure, and memory swapping rarely happens. This also implies that the encryption overhead for inputs and outputs is negligible. However, as the request rate increases, the latency of the CC-enabled GPU grows significantly. This is due to the occurrence of intermediate data swapping. Taking a closer look, we discover that the bottleneck primarily stems from memory "swapping in": when the GPU requests data, the CPU must encrypt the data before sending it to the GPU. However, during the CPU encryption phase, the GPU is idle due to the unavailability of the input. These idle cycles affect the latency of the ongoing request and other pending requests, thus exacerbating the overall latency overhead.

From these experiments, we summarize that when handling multiple requests, serving even mid-sized LLMs with CC-enabled GPUs imposes significant overhead. The main cause of this overhead is the memory pressure caused by the intermediate data (in particular, KV cache) of multiple requests. Furthermore, we note that the main bottleneck is the CPU encryption when "swapping in".

Case study 3: fine-tuning LLMs. Fine-tuning is a widely used technique for customizing LLMs. In this experiment, we aim to understand the performance bottlenecks of fine-tuning LLMs in a confidential computing environment. We experiment with Parameter-Efficient Fine-Tuning (PEFT) [31] with DeepSpeed [39], to evaluate the overhead brought by the CC-enabled environment. Our experiments run LoRA [16] on OPT-30B and OPT-13B, using the ultrachat dataset [30], with the maximum batch size to trigger memory swaps.

Figure 3c shows the result of PEFT. The throughput drop brought by confidential computing is 36.2% on OPT-30B model and 14.0% on OPT-13B model. The overhead is smaller on OPT-13B because it contains fewer parameters than OPT-30B and has lower memory pressure, requiring less I/O. Overall, the performance degradation stems from model offloading, similar to FlexGen.

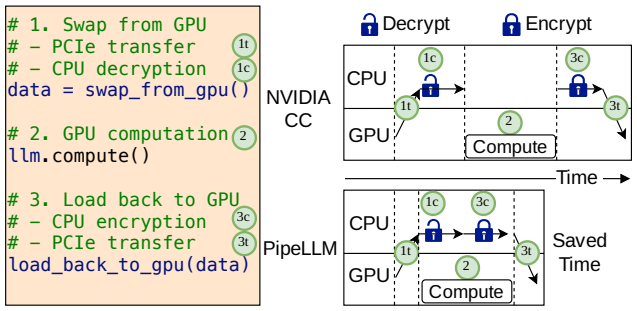
4 Speculative Pipelined Encryption

In this section, we describe our main idea, the problem statement, and technical challenges to build our system, PipeLLM. The primary goal of PipeLLM is to minimize the overhead of NVIDIA Confidential Computing, while preserving its (1) security guarantees and (2) user transparency. By user transparency, we mean that PipeLLM applies to non-modified LLM applications, including LLM models, deep learning frameworks, and any other supporting code and data. Next, we elaborate on PipeLLM's threat model.

Threat model. NVIDIA Confidential Computing aims at protecting the confidentiality and integrity of applications running on GPUs; for LLM applications, these are the model weights and user request and response tokens. PipeLLM shares the same threat model as NVIDIA Confidential Computing: attackers can have full control over the hypervisor and the host OS, but they cannot access the private memory protected by confidential VMs and CC-enabled GPUs. Like confidential computing on CPUs [2, 19], NVIDIA Confidential Computing is susceptible to DOS attacks and side channel attacks; PipeLLM shares the same limitation. In fact, PipeLLM introduces new side channels, discussed in §8.1.

4.1 Main idea

As mentioned in §3, the bottleneck when serving and fine-tuning confidential LLMs is the encryption of memory swapping. Our main idea is to somehow remove the encryption out of the critical path of memory swapping. Note that encryption is fundamental for security hence cannot be avoided; it can only be hidden. Hiding computation by pipelining is a well-established approach. We plan to pipeline encryptions, data transfers, and GPU computation for better performance. Figure below visualizes our main idea along with a comparison to NVIDIA Confidential Computing. On the left is an example of toy code; on the right is an illustration of how NVIDIA Confidential Computing and PipeLLM execute it.



For transparency, NVIDIA Confidential Computing performs on-the-fly encryption and decryption (indicated by "1c" and "3c") within its interface (e.g., `cudaMemcpyAsync`), together with data transfer steps ("1t" and "3t"). PipeLLM decouples and pipelines data transfer, encryption, and decryption to minimize GPU idle time. Note that GPU-side encryption and decryption are not depicted.

A major challenge to realize PipeLLM's pipelining is pre-encrypting data (i.e., the "3c"). PipeLLM must "predict" which data the GPU will request and, crucially, the precise order of these requests. This sequence is essential due to cryptographic requirements, which we elaborate on below.

Swapping sequence and IVs. As introduced in §2.2, NVIDIA Confidential Computing (H100) encrypts I/Os using AES-GCM, a stream cipher that requires an *IV* which is an integer. It is crucial that the IV for each encryption differs from the previous ones. To pipeline encryption and data transfer, one needs to somehow guess the sequence of memory swapping a priori and pair the sequence with the increasing IVs. We focus on the scenario where the CPU acts as the encryptor. GPU-to-CPU data transfer is not part of the prediction process, so its IV does not need to be predicted. Next, we define the memory-copy prediction problem.

4.2 Problem Statement

For an LLM application, when swapping memory (e.g., model weights and KV cache), the LLM system submits a sequence of memory-copy commands (`cudaMemcpyAsync`); it then uses a synchronization API (`cudaDeviceSynchronize`) to ensure the data has been loaded into the GPU's memory; finally, it starts computation on the GPU. Note that other than memory swapping, there are memory-copies between CPUs and GPUs for normal computations.

The problem to solve is to predict the next several memory copies and their IVs, given the current IV and the execution context. Since there is no modification to the LLM applications, only low-level memory-copy information is available. We assume LLM models are known.

Predicting GPU memory copies in general is hard. For serving and fine-tuning LLMs, we want to ignore normal memory copies and *only* predict memory swaps because (i) memory swaps contribute to the most of encrypted data transfers and are the performance bottleneck (§3); (ii) the swapped memory remains unchanged on CPU, hence can be encrypted ahead of time without worrying the contents will be modified; (iii) LLM inference and fine-tuning are highly regular (§2.1); thus, accurately predicting memory swapping sequence is possible.

Observations. To predict memory swaps, we observe that the sizes of LLM memory copies have the following patterns: (1) the size of memory swapping (usually >128KB) is significantly larger than other data transfers (usually <8KB). (2) we can distinguish model offloading and KV cache swapping by calculating their sizes ahead of time based on the target LLMs. The two patterns allow us to accurately classify memory copies. In addition, most of the transferred data is memory swaps which are read-only. This enables an efficient validation by leveraging page faults (we describe details below, §5.2). Finally, we observe that performing a "NOP" data transfer (a 1-byte dummy) increments the current IV, with low overhead. This implies that predicting a larger IV than the ground truth has little penalty.

4.3 Speculative Pipelined Encryption

At a high level, PipeLLM's prediction policy repeats the swapping patterns it has seen before, and predicts swaps with slightly larger IVs as a leeway for compensating other data transfers. For example, FlexGen almost always swaps transformer layers in order. So, when transferring layer i, PipeLLM predicts and encrypts layer $i + 1$. As another example, vLLM uses FIFO (First-In, First-Out) and LIFO (Least-In, First-Out) to swap KV cache for different chunks. PipeLLM will predict swaps by FIFO and LIFO accordingly.

With the above prediction, PipeLLM introduces *speculative pipelined encryption* to *predict* a sequence of swapped memory chucks in the near future and *pipeline* the encryptions and ciphertext transfers to hide the bottleneck. Furthermore, PipeLLM develops a set of techniques to lower the penalty of incorrect predictions. The speculative pipelined encryption can be divided into three stages:

1. *Prediction stage*: in this stage, PipeLLM detects which GPU memory belongs to model weights and KV cache without looking into the application's code. It further predicts the sequence of memory chunks to be swapped, and encrypts them with corresponding IVs.
2. *Validation stage*: at the moment of the application submitting the memory-copy commands, PipeLLM validates the pre-encrypted data by checking if the data is correctly encrypted (e.g., no update to the plaintext) and if using the right IVs. If the check fails, PipeLLM goes to the error-handling stage.
3. *Error-handling stage*: when the requested sequence is different from the predicted sequence, PipeLLM needs to relinquish the current pipeline and start with the ground-truth sequence.

Technical challenges. PipeLLM faces multiple technical challenges. We list them below and address them in the next section (§5). First, PipeLLM has no knowledge about high-level semantics (e.g., which piece of data to swap), which significantly helps predictions. However, getting them requires modification to the application and violates the user-transparency goal (§4). Therefore, PipeLLM has to make predictions based on limited low-level information, including LLM features and API trace of applications.

Second, PipeLLM needs a computational and memory efficient validation. Validation checks whether the pre-encrypted data aligns with what is requested. The challenge is how to efficiently check if the data has been updated. Again, note that PipeLLM has no information about whether the data is read-only (e.g., swapped data) or read-write (e.g., data requiring CPU updates). One straightforward solution is to store (a) the plaintext (that might be updated), (b) the pre-encrypted ciphertext, and (c) the original plaintext associated with the pre-encrypted ciphertext. By comparing (a) and (c), PipeLLM will know if (b) is valid. It however requires data comparison on the critical path, which increases runtime overhead. This solution also triples the memory usage.

Third, PipeLLM needs to handle prediction errors efficiently. Pipeline relinquishing is expensive. On the one hand, when encountering an error, the wanted data needs to be encrypted on-the-fly, leading to high tail latency. On the other hand, the encrypted data remaining in the pipeline cannot be used due to incorrect IVs, which requires another round of encryption for all the subsequent data.

5 PipeLLM Design

PipeLLM architecture. We show the architecture of PipeLLM in Figure 4. PipeLLM works on the library and driver level, requiring no modification to user applications. PipeLLM consists of a predictor (§5.1), a validator (§5.2), and an error handler (§5.3). The predictor produces the next sequence of memory chunks to swap, based on the low-level information observed by PipeLLM; the validator checks if the pre-encrypted data is valid before sending it to the GPU; the error handler re-plans the pipeline when facing a prediction error.

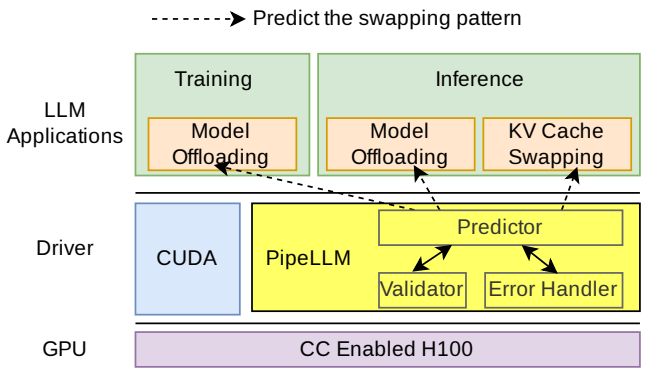


Figure 4: PipeLLM's architecture. Yellow parts are PipeLLM.

5.1 PipeLLM Predictor

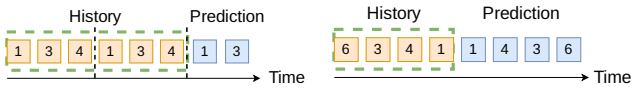
PipeLLM's predictor predicts which memory chucks to be swapped in and the chunks' corresponding IVs. In practice, the swapping requests are in batch: a set of memory copy operations (i.e., `cudaMemcpyAsync`) followed by a synchronization (i.e., `cudaDeviceSynchronize`). The LLM systems such as FlexGen, vLLM, and PEFT need to maintain the order between batches, without enforcing a specific order within a batch.

The predictor takes in the following three inputs: (1) a swapped-in batch history $[B_0, \cdots, B_n]$, in which each B_i contains a set of memory chucks $\{C_p, \cdots, C_q\}$ and B_n is the most recent swapped-in batch; (2) the swapped out memory blocks $\{C_i, \cdots, C_j\}$ on main memory which haven't been swapped into the GPU yet, and (3) the current Initialization Vector (IV), IV_{cur}. The predictor can be abstracted as a function f that predicts the next swapped-in block C_{next} and its corresponding IV_{next}:

$$f([B_0, \cdots, B_n], \{C_i, \cdots, C_j\}, IV_{cur}) \rightarrow (C_{next}, IV_{next})$$

So far, there is a limited number of swapping patterns in today's systems, including repetitive pattern (e.g., model weights offloading), FIFO (e.g., layer-wise KV cache swapping), and LIFO (e.g., request-wise KV cache swapping). We elaborate on these patterns below. However, PipeLLM's predictor is general and can easily extend to other patterns. To implement a new pattern, one needs to recognize the pattern from the history and write a prediction function given the current swapping states. Our future work is to use ML models to learn the predictor f without human efforts.

Current swapping patterns. In the following, we describe several swapping patterns we observed in today's LLM systems. For model offloading like FlexGen, it has a repetitive



(a) Repetitive pattern **(b)** LIFO (last-in-first-out)

Figure 5: Common patterns in LLM memory swapping.

pattern. Figure 5a gives an example. Each box represents a layer, tagged by the layer number in the model (e.g, the first layer is tagged by 1). Parameters from layer 1 are first reloaded, followed by layers 3 and 4, and then back to layer 1. This pattern indicates that layers 1, 3, and 4 are offloaded. Given that the most recent layer in the swap history is layer 1 (the final red block in the Figure 5a), PipeLLM predicts that layer 3 will be the next to be reloaded and prepares it with an appropriate IV for encryption. We observe that model offloading in most applications typically follows this repetitive pattern. Since each layer in LLMs shares the same structure and number of parameters, it is unlikely that applications offload a particular layer in some iterations while retaining it in the GPU during other iterations.

Another major swapping in LLMs is KV cache swapping, typically conducted on a layer-wise or request-wise basis. In layer-wise swapping, the application decides whether to swap the KV cache generated in each layer for all requests in the batch. Alternatively, request-wise approach involves swapping all layers of KV cache from certain requests in case of memory shortage to free up memory for others. Layer-wise swapping aims to achieve high throughput as it allows the batch size to remain unchanged; request-wise swapping reduces latency for high-priority requests and lowers the average latency overall.

In layer-wise swapping, applications swap out KV cache of each layer in order, and then retrieve them in the same order, thus the pattern is FIFO. In request-wise swapping, applications usually swap out the lowest priority request in the current batch when facing memory shortage. Therefore, the first request to be swapped out is likely the one with the lowest priority among the requests, and it will be the last to be reloaded. So the swapped in pattern is LIFO, as depicted in Figure 5b.

Targeting major memory swapping. The predictor focuses on major GPU memory swaps—model weights offloading and KV cache swapping. Other than the two, I/Os also involve other small data transfers, for example, user input and output tokens. As mentioned in §3, since the encryption of them bring negligible overheads, PipeLLM does not pipeline them. By the model definition (i.e., its computational graph and weights), PipeLLM distinguishes the data of model weights and KV cache with high accuracy without additional hints from LLM applications. Therefore, PipeLLM can distinguish the data of model weights and KV cache with high accuracy without additional hints from LLM applications. Therefore, PipeLLM knows if a memory swap is model offloading or KV cache swapping with high confidence.

Notably, those small data transfers would make the IV increment. Since swapping and small I/O are intertwined and interleaved in applications, PipeLLM would predict a larger IV for swapping to handle IV increments by small I/O. If predicted IV is larger, the ciphertext could still be utilized for transfer, which would be explained in §5.3.

5.2 Validator: Validating the Consistency of the Ciphertext

The problem. As mentioned, PipeLLM speculatively encrypts memory chunks that haven't been requested by the GPU. The problem however is, how does PipeLLM know if the original data has been updated when later requested? For example, consider the case that an application transfers some data from the GPU to the main memory, updates the memory in-place, and then writes it back to the GPU. In this case, this method could result in sending ciphertext of outdated data when speculative encryption is performed before data update by the application. Therefore, before sending ciphertext, PipeLLM must verify that the corresponding plaintext matches the expected data.

Our method. PipeLLM's validator uses an efficient approach to validate the encrypted data. The validation works as follows. During the prediction stage, PipeLLM labels each pre-encrypted ciphertext with the corresponding plaintext's address range on the CPU memory. PipeLLM then revokes the write permission of these memory pages. Therefore, when facing data modifications by the application, PipeLLM detects the changes by a page fault. The page protection can thus prevent PipeLLM from violating the system's correctness. During the validation stage at runtime, PipeLLM checks if the address and length of the swap align with the associated labels to determine the match between plaintext and ciphertext. This method is highly effective for LLMs because applications do not move or update the swapped model weights or KV cache in the CPU memory. The pre-encrypted ciphertext only involves large memory transfers which are typically non-modified swaps. Therefore, page faults rarely occur and thus bring little performance overhead.

5.3 Error Handler: Tolerating Wrong IVs with Re-Ordering and NOP Padding

PipeLLM may incorrectly predict the swapped memory chunks, their sequence, or their IVs. Some errors are "irrecoverable": for example, if the predicted IV is smaller than the current IV, the pre-encrypted data must be discarded. In such cases, PipeLLM's error handler resets the pipeline by discarding the speculative encrypted memory and restarting the process. However, many errors are recoverable and can be corrected with specific modifications to maintain pipeline integrity.

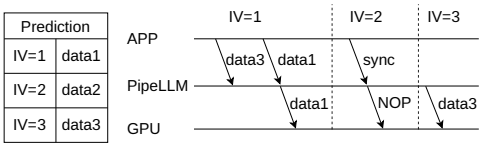


Figure 6: Tolerate wrong IVs with re-ordering requests and padding NOPs. In this figure, the current IV starts with 1, and the application requests "data3" which is speculatively encrypted by IV=3. Since IV=3 is greater than the current IV=1, PipeLLM suspends this request. The next request is "data1" which is encrypted by IV=1, so PipeLLM sends "data1" and IV advances to 2. Finally, by synchronization "sync", PipeLLM sends a NOP to increase the current IV=3, invalidates the incorrectly encrypted memory "data2", and commits the pending requests "data1" with IV=3.

We introduce two approaches—swap re-ordering and NOP padding—to address these errors. We experiment with various incorrect predictions in § 7.3.

Swap re-ordering. Swap re-ordering helps the case that PipeLLM incorrectly predicts IV of memory blocks one the batch, since each block within the batch is encrypted with a unique IV. As mentioned in § 5.1, LLM applications may batch multiple swap-in memory blocks, in which a synchronization (e.g., `cudadevicesynchronize`) marks the boundary of each batch. PipeLLM can re-order these speculative encrypted blocks within the batch, making the memory blocks swapped at predicted IV without having to relinquish the pipeline.

Padding NOPs. A NOP is an operation to send a 1-byte piece of data to increase the current IV. Padding NOPs is useful when PipeLLM encrypts a data block with an IV that is larger than it should be. For example, in Figure 6, "data3" is speculatively encrypted using IV=3; however, current IV=2. PipeLLM can forward IV by sending a NOP. In addition, NOP padding enables PipeLLM to tolerate small-sized data transfers (mentioned in §5.1) without restarting the pipeline.

5.4 Asynchronous Decryption

So far, our discussion has primarily focused on CPU-to-GPU data transfers and CPU-side encryption. Regarding CPU-side decryption during GPU-to-CPU data transfers, we observe that by default, decryption is unnecessarily synchronous. In particular, LLM applications do not alter swapped-out model weights or KV cache. Based on this observation, PipeLLM implements asynchronous decryption for swapped-out data. The memory copy returns before decryption and leaves the data at the destination address unmodified.

As a technicality, the async decryption works as follows. PipeLLM determines whether the data being transferred is part of the KV cache or model parameters based on their feature on data size mentioned earlier. If yes, PipeLLM decrypts it asynchronously for better overlapping; otherwise, PipeLLM does the default, decrypting the data immediately. At misprediction, the applications may process the data before decryption. To detect usage-before-decryption, PipeLLM revokes the read and write permissions of the application for the plaintext which is a placeholder before decryption finishes. If the application attempts to access (read or write) this data, a page fault occurs. PipeLLM then decrypts the data synchronously and allows the application to continue its execution.

6 Implementation

Current CUDA uses a zero-copy technique for ciphertext in CVM shared memory: for example, on host-to-device memory copy, the plaintext is stored in CVM private memory before encryption; the encryption loads the plaintext and directly writes the ciphertext in CVM shared memory. In our design, employing a zero-copy technique could expose uncommitted ciphertext, posing a potential security risk. To mitigate this, our system (PipeLLM) stores the predicted ciphertext in CVM private memory and only transfers it to shared memory after validating the prediction. Therefore, unvalidated ciphertext is not exposed in CVM shared memory to attackers and the security is not compromised. To eliminate the memory copy overhead, PipeLLM leverages fix-sized buffers in shared memory for DMA to eliminate memory allocation from the critical path and pipelines data copy with PCIe transfer. Because memory copy is faster than PCIe, the pipeline does not require much buffer for the memory copy stage, minimizing the usage of CVM shared memory.

We implement a prototype of PipeLLM as user-space runtime. We hack CUDA APIs related to CPU-GPU data transfer (e.g., `cudaMemcpyAsync`) to implement the pipelining. Although CUDA implements the coupled memory copy API in proprietary code, it calls OpenSSL APIs (`EVP_EncryptUpdate`) encryption and decryption. PipeLLM also hacks those OpenSSL APIs to decouple encryption or decryption from the memory copy API. To decouple encryption from host-to-device copy, PipeLLM copies the pre-encrypted ciphertext to the destination buffer provided by CUDA. For decryption, the corresponding API called by CUDA would return without performing the decryption. which is pipelined later. PipeLLM uses MPK/PKU [20] to implement the read/write access revoke with little overhead, which can apply to Intel TDX and AMD SEV-SNP. The core logic of PipeLLM consists of 958 lines of C++ code, with another about 600 lines of code for hacking the APIs.

7 Evaluation

We answer the following questions in the evaluation section:

1. What degree of improvement does PipeLLM bring to the performance of LLM inference and fine-tuning systems in confidential computing?
2. To what extent does the pipelining contribute to the overall performance improvement?

3. Does PipeLLM's performance rely on the high accuracy of predictions?

7.1 Evaluation Setup

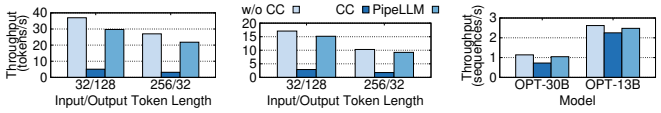
Platform configurations. We setup a KVM VM on an Intel server equipped with dual Intel Xeon Platinum 8462Y+ CPUs at 4.10 GHz. This VM is configured with 16 virtual CPUs (vCPUs) and 250GB of memory, running Ubuntu 20.04 LTS. Additionally, a single H100-SXM GPU, connected via a PCIe 5.0 link capable of a maximum duplex bandwidth of 128GB/s, is integrated into the server. This GPU is directly passed through to the VM for dedicated use. In our experiments, the GPU enables confidential computing, while the VM only enables encryption for I/O between GPU and CPU, as our CPUs do not support Intel TDX.

Workloads and metrics. We conducted experiments on the cutting-edge inference systems, FlexGen [42], PEFT [31] and vLLM [25], under confidential computing environment. These LLM applications utilize model offloading and KV cache swapping to address GPU memory shortage. FlexGen is a leading LLM inference system renowned for its model swapping capabilities in high-throughput scenarios. PEFT is the state-of-the-art LLM training system with support of LoRA fine-tuning and integration of DeepSpeed [39] model offloading. vLLM, the state-of-the-art LLM inference system in serving scenarios, uses KV cache swapping to handle the GPU memory shortage. We utilize FlexGen to assess the performance of model offloading, with synthetic datasets to evaluate inference throughput. In each test case of FlexGen, 1000 requests are generated and we report the average throughput. We run PEFT fine-tuning using ultrachat [30] dataset for one epoch (about 6k sequences) and report the training throughput. Additionally, we used vLLM to evaluate performance in KV cache swapping, utilizing ShareGPT [47] and Alpaca [46] to cover a range of request lengths. vLLM's focus is on serving, hence we evaluated normalized latency across varying request rates to demonstrate its serving capabilities. In each test case of vLLM, 30-minute traces are used to evaluate the systems.

Baselines. We conducted a comparative analysis of PipeLLM against two baseline systems. The first baseline, referred to as "w/o CC" (without Confidential Computing), represents the native performance of inference systems, without the confidential computing feature. The second baseline, denoted as "CC" (Confidential Computing), showcases the performance of the current NVIDIA Confidential Computing framework specifically applied in LLM inference.

7.2 End-to-End Performance

We demonstrate that PipeLLM effectively reduces overhead in various swapping scenarios, encompassing model offloading and KV cache swapping. Remarkably, this reduction in overhead is achieved without compromising security. Furthermore, the performance enhancements offered by PipeLLM are not confined to a particular model or configuration; they are broadly applicable, underscoring the system's versatility and effectiveness across different setups.



(a) FlexGen OPT-66B **(b)** FlexGen OPT-175B **(c)** PEFT

Figure 7: Performance of FlexGen and PEFT with model offloading.

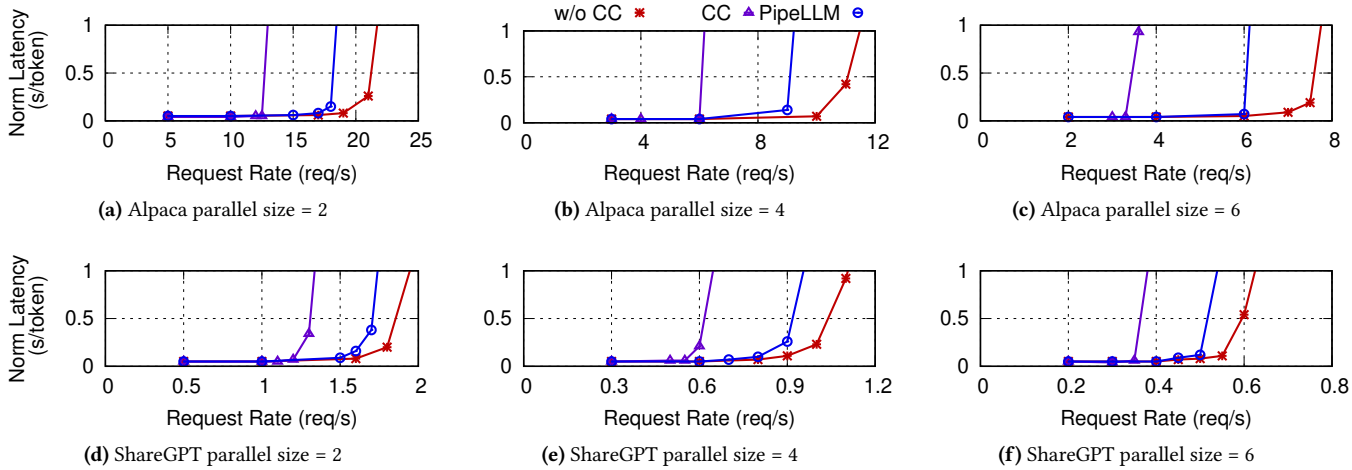
Model offloading. We conducted tests on model offloading using FlexGen as inference cases and PEFT[31] as fine-tuning cases. For FlexGen, our evaluation focused on two specific models: OPT-66B and a 4-bit-quantized version of OPT-175B. We use multiple configurations for running FlexGen. The input token length was set to either 32 or 256, and the output was set to either 128 or 32. For PEFT, OPT-30B and OPT-13B are used for LoRA-based fine-tuning. The memory requirements of these models exceed the capacity of a single GPU. To evaluate model offloading, we maintained the KV cache and runtime temporary data on the GPU, limiting the offloaded data exclusively to model weights.

A key aspect to note is that the performance of model offloading is bound by PCIe throughput. For example, the peak swapping throughput of "w/o CC" in FlexGen is about 56GB/s. To maintain optimal performance, it is imperative for PipeLLM to generate ciphertext at a speed that matches or exceeds the swapping throughput. Therefore, PipeLLM would utilize multiple CPU threads dedicated to encryption and decryption to optimize model offloading.

Figure 7 presents the PipeLLM's performance in running FlexGen and PEFT in the context of model swapping with various models and input configurations. Enabling confidential computing typically results in a substantial performance decline, ranging from 82.8% to 88.2% in FlexGen, and up to 36.2% in PEFT.

PipeLLM successfully mitigates this performance degradation, reducing the overhead to less than 19.6%. It is a great improvement and highlights the efficacy of PipeLLM in maintaining system performance even under the constraints of confidential computing. The remaining overhead mainly owes to 40GB/s maximum bandwidth of CPU-to-GPU memory copy even if all overhead of CPU-side encryption is eliminated.

KV cache swapping. We conducted tests with vLLM, the state-of-the-art LLM inference system in serving scenarios. Our evaluation focused on two models: OPT-30B and OPT-13B. During these evaluations, we ensured that all model weights for these models were stored in the GPU. Our experimental approach involved multiple configurations. We set



(a) Alpaca parallel size = 2 **(b)** Alpaca parallel size = 4 **(c)** Alpaca parallel size = 6

(d) ShareGPT parallel size = 2 **(e)** ShareGPT parallel size = 4 **(f)** ShareGPT parallel size = 6

Figure 8: Performance on vLLM OPT-30B model with KV cache swapping.

the parameter of parallel sampling to 2, 4, and 6, in consistency with the methodologies outlined in the original paper. For vLLM, PipeLLM only uses one thread for encryption and one thread for decryption.

Figure 8 provides a comprehensive analysis of PipeLLM's performance in handling KV cache swapping for the OPT-30B model. The data reveals that confidential computing's encryption and decryption results in an overall performance drop, ranging from 33.3% to 52.8%. PipeLLM reduces the overhead to 5.2% - 14.2%. Although the prediction success rate is near 100%, PipeLLM is not able to reduce the overhead of decreased throughput of data transfer (from 64GB/s to about 40GB/s) as well as the control-plane overhead of NVIDIA Confidential Computing (as analyzed in §3).

The observed performance degradation of "CC" with the OPT-13B model in ShareGPT dataset is 15.3% - 23.6%, and less than 8% overhead in Alpaca dataset, notably lower than that experienced with the OPT-30B model, owing to the distinct memory usage characteristics of the two models. Specifically, OPT-13B's model weights occupy only about 26GB, which is 32.5% of the GPU memory, in contrast to OPT-30B, which utilizes a significantly larger portion, approximately 60GB or 75% of the GPU memory. However, PipeLLM can still reduce the overhead to less than 8% in OPT-13B cases.

7.3 Analysis of Pipelining

PipeLLM introduces pipelining to enhance performance, utilizing multi-threading to boost encryption throughput. It is important to note that "CC" can also use multiple CPU threads to attain a line-rate encryption rate on the CPU, thereby narrowing the performance gap. Our findings demonstrate that "CC" requires a greater number of threads compared to PipeLLM to achieve comparable performance.

We conducted the experiment using the vLLM with the OPT-30B model. The dataset employed was Alpaca, and the

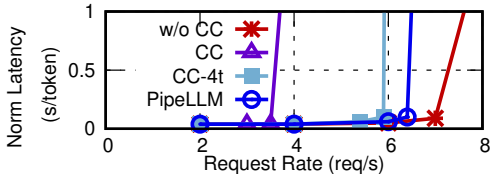


Figure 9: Performance of trivial multi threading on vLLM OPT-30B. "CC-4t" refers to the system using 4 threads for encryption and decryption without pipelining.

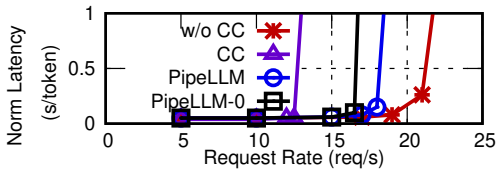


Figure 10: Ablation study on success rate. "PipeLLM-0" refers to the system with zero success rate on sequence prediction.

parallel size set for the experiment was 6. Figure 9 illustrates the performance comparison between "CC", utilizing four threads for encryption and decryption, and the alternatives 'w/o CC' and PipeLLM. Notably, PipeLLM only uses two threads and yet outperforms "CC" with four threads but in the absence of pipelining. This highlights the effectiveness of pipelining in this context.

7.4 Ablation Study on Success Rate

PipeLLM achieves near 100% success rate on KV cache swapping in vLLM, because vLLM takes LIFO as its swap policy, bringing benefit to PipeLLM. To show that PipeLLM works well on inference systems with other swap policies on KV cache, we conduct experiments on prediction failure on the sequence.

We use vLLM with the OPT-30B model, Alpaca dataset, and a parallel size of 2 as an example. Figure 10 shows that the performance of PipeLLM with zero prediction success rate only slightly drops by 8.3%, mainly caused by the overhead of NOPs. Upon sequence prediction failure, PipeLLM can still use the ready ciphertext and use NOP to drop the mispredicted ciphertext. The additional encryption latency is well hidden with GPU computation.

8 Discussion

8.1 Security Analysis

PipeLLM preserves the confidentiality and integrity features of NVIDIA Confidential Computing. The MPK/PKU mechanism used for validating predictions in PipeLLM operates within the guest VM and is secured from attackers by CVM hardware. Importantly, in PipeLLM's design, no cryptographic data is transferred to shared memory until a prediction is verified. Additionally, the padding NOPs described in §5.3 contain dummy data, preventing leakage of information to attackers. Consequently, the ciphertext of any mispredicted transfer does not compromise the security guarantees of NVIDIA Confidential Computing.

Although PipeLLM does not compromise confidentiality or integrity, its mis-speculation introduces side channels in NOP transfers compared to NVIDIA Confidential Computing, including (1) attackers can detect if the LLM system is currently swapping by observing NOPs, and (2) attackers could profile the frequency of prediction failures, potentially revealing the swapping patterns of applications. The security implications of these side channels remain unclear and need further research.

8.2 Should Swap Data Be Re-encrypted?

As discussed in §3, the primary overhead of NVIDIA Confidential Computing in LLM workloads arises from encrypting swap data. Observing that applications on the CPU do not modify this data, one possible approach is to retain the encrypted version on the CPU. This would allow the encrypted data to be transferred directly to the GPU during loading, thereby eliminating encryption overhead. While this approach improves performance, implementing it naively compromises security guarantees. For example, reusing encrypted data enables attackers to identify data that matches a previous transfer; more critically, it could make the system vulnerable to replay attacks [35].

Currently, NVIDIA Confidential Computing uses the AES-GCM algorithm with incrementing IVs to prevent ciphertext reuse and defend against replay attacks (§ 2.2). This approach prioritizes simplicity and transparency at the cost of performance: it encrypts read-only data on-the-fly, incurring some overheads, but it requires no modifications to applications or encryption engines. PipeLLM follows this design choice, providing practical performance benefits while preserving transparency for users.

However, this design is a trade-off rather than a fundamental limitation. Future confidential GPUs could benefit from dedicated hardware and library interfaces to support ciphertext reuse for swap data, potentially alleviating the bottlenecks faced by today's CC-enabled LLM systems.

8.3 Compared with TEE I/O Approach

The overhead of CC-enabled LLMs could also by mitigated by hardware encryption. The next generation of CVM introduces TEE I/O [21], equipped with dedicated hardware on CPU SOC for line-rate encryption. However, limited performance information is currently available. In practice, a GPU server typically runs multiple VMs—a standard H100 server, for example, has two CPUs (dual-socket) and eight GPUs—raising questions about whether the TEE I/O hardware can sustain GPUs' throughputs. Compared to hardware solutions, PipeLLM offers greater flexibility for different hardware configurations.

9 Related Work

Enclaves that target CPU Various commercial products have been released to provide CPU enclaves for protecting user data and code from untrusted operating systems and hypervisors. An enclave has isolated CPU states and memory region protected by the hardware, so that the CPU and memory data cannot be directly accessed by the privileged software. Intel SGX [6] is the first enclave implementation that provides a process-level enclave, which mandates the application to be modified to use the SGX interface.

The CVM abstraction, including AMD SEV [1, 2], Intel TDX [17, 19] and ARM CCA [3], enables a more application-transparent approach by running the user workloads in VM user mode without modifications [8, 9, 27, 28, 32]. However, for the machine learning workloads, especially LLMs, these CPU-side enclaves should be combined with GPU-side enclave to provide end-to-end protection for them.

GPU Enclave In addition to software-based GPU enclaves, researchers have also explored hardware-based approaches. Graviton [50] represents the pioneering effort in creating a GPU enclave, achieved by modifying the GPU's command processor and using data encryption to safeguard data confidentiality. HIX [22], without altering GPU hardware, proposes modest extensions to the I/O interconnect and the memory management unit (MMU) to secure GPU computations. Another innovative approach, HETEE [54], leverages PCIe switch fabric to control access to GPUs securely and flexibly without necessitating hardware modifications. StrongBox [13] introduces an integrated-GPU enclave for ARM devices, utilizing the pre-existing TrustZone features. Although this paper focuses specifically on the design of

PipeLLM for the NVIDIA H100 GPU, the design considerations are equally applicable to other GPU enclaves.

GPU Attacks. Several studies have scrutinized the exploitation of security vulnerabilities to compromise GPU isolation. A notable vulnerability involves the potential breach of GPU security, allowing for the execution of GPU-based malware. This malware can circumvent IOMMU protections and illicitly access private information from the host CPU [55]. Another common avenue of attack targets residual memory contents left on the GPU from previous computations. For instance, Lee et al.[26] demonstrated methods of extracting sensitive webpage data retained in GPU memory, particularly from Chromium and Firefox browsers. Similarly, CUDA Leaks[37] revealed the possibility of extracting sensitive information, including plaintext and encryption keys, from the GPU's shared and global memory.

LLM-Specific Optimizations. Extensive research has been conducted to enhance the performance of LLM without CC, such as operator-level optimizations [11, 12], efficient memory management on GPU [25, 41], and optimizations based on sparsity [44, 45, 51]. PipeLLM is orthogonal to these optimizations if running within CC.

10 Conclusion

This paper comprehensively evaluates and analyzes the performance overhead when serving and fine-tuning LLMs on CVM equipped with a GPU enclave. This paper introduces PipeLLM, a user-transparent runtime system that effectively reduces the performance overhead associated with GPU-based confidential computing. PipeLLM uses speculative pipelined encryption to overlap encryption and data transmission, thereby significantly minimizing latency. Our performance evaluations with real-world LLM systems, vLLM, PEFT, and FlexGen, demonstrate that PipeLLM can reduce overhead to <19.6% across various LLM sizes.

Acknowledgments

We express our sincere gratitude to our shepherd Kiwan Maeng, and the anonymous reviewers for their insightful comments. This work was partially supported by NSFC (No. 62372287 and 61925206). Cheng Tan is supported in part by NSF CAREER Awards #2237295. Zeyu Mi (yzmizeyu@sjtu.edu.cn) is the corresponding author.

References

[1] AMD. Protecting VM Register State With SEV-ES. https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf, 2017.

[2] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020.

[3] ARM. ARM Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, 2023.

[4] Microsoft Azure. Announcing Azure confidential VMs with NVIDIA H100 Tensor Core GPUs in Preview. https://aka.ms/cvm-h100-preview, 2023.

[5] Microsoft Azure. Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/, 2024.

[6] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, October 2014. USENIX Association.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[8] Jiahao Chen, Dingji Li, Zeyu Mi, Yuxuan Liu, Binyu Zang, Haibing Guan, and Haibo Chen. Security and performance in the delegated user-level virtualization. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 209–226, Boston, MA, July 2023. USENIX Association.

[9] Jiahao Chen, Zeyu Mi, Yubin Xia, Haibing Guan, and Haibo Chen. CPC: Flexible, secure, and efficient CVM maintenance with confidential procedure calls. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 1065–1082, Santa Clara, CA, July 2024. USENIX Association.

[10] Google Cloud. Oh SNP! VMs get even more confidential. https://cloud.google.com/blog/products/identity-security/rsa-snp-vm-more-confidential, 2024.

[11] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[13] Yunjie Deng, Chenxu Wang, Shunchang Yu, Shiqing Liu, Zhenyu Ning, Kevin Leach, Jin Li, Shoumeng Yan, Zhengyu He, Jiannong Cao, and Fengwei Zhang. Strongbox: A gpu tee on arm endpoints. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 769–783, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Gobikrishna Dhanuskodi, Sudeshna Guha, Vidhya Krishnan, Aruna Manjunatha, Rob Nertney, Michael O'Connor, and Phil Rogers. Creating the first confidential gpus. *Commun. ACM*, dec 2023.

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[17] Intel. Intel TDX® Module v1.5 Base Architecture Specification. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-module-1.5-base-spec-348549001.pdf, 2022.

[18] Intel. Intel® Trust Domain Extension (Intel® TDX) Module. https://www.intel.com/content/www/us/en/download/738875/intel-trust-domain-extension-intel-tdx-module.html, 2022.

[19] Intel. Intel® Trust Domain Extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, 2022.

[20] Intel. Intel software developer's manual. https://cdrdv2-public.intel.com/812392/325462-sdm-vol-1-2abcd-3abcd-4.pdf, 2023.

[21] Intel. Intel® TDX Connect Architecture Specification. https://cdrdv2.intel.com/v1/dl/getContent/773614, 2023.

[22] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 455–468, New York, NY, USA, 2019. Association for Computing Machinery.

[23] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[24] David Kaplan. Hardware vm isolation in the cloud. *Commun. ACM*, 67(1):54–59, dec 2023.

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[26] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In *2014 IEEE Symposium on Security and Privacy*, pages 19–33, 2014.

[27] Dingji Li, Zeyu Mi, Chenhui Ji, Yifan Tan, Binyu Zang, Haibing Guan, and Haibo Chen. Bifrost: Analysis and optimization of network I/O tax in confidential virtual machines. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 1–15, Boston, MA, July 2023. USENIX Association.

[28] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. Twinvisor: Hardware-isolated confidential virtual machines for arm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 638–654, New York, NY, USA, 2021. Association for Computing Machinery.

[29] HaoHui Mai, Jiacheng Zhao, Hongren Zheng, Yiyang Zhao, Zibin Liu, Mingyu Gao, Cong Wang, Huimin Cui, Xiaobing Feng, and Christos Kozyrakis. Honeycomb: Secure and efficient GPU executions via static validation. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 155–172, Boston, MA, July 2023. USENIX Association.

[30] Sourab Mangrulkar. ultrachat-10k-chatml. https://huggingface.co/datasets/smangrul/ultrachat-10k-chatml, 2024.

[31] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

[32] Zeyu Mi, Dingji Li, Haibo Chen, Binyu Zang, and Haibing Guan. (mostly) exitless VM protection from untrusted hypervisor through disaggregated nested virtualization. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1695–1712. USENIX Association, August 2020.

[33] Microsoft. Microsoft Copilot: Your everyday AI companion. https://copilot.microsoft.com/, 2024.

[34] NVIDIA. Benefits of NVIDIA Hopper H100 Confidential Computing for trustworthy AI. https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/, 2023.

[35] NVIDIA. Confidential Compute on NVIDIA Hopper H100. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf, 2023.

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

[37] Roberto Di Pietro, Flavio Lombardi, and Antonio Villani. Cuda leaks: A detailed hack for cuda and a (partial) fix. *ACM Trans. Embed. Comput. Syst.*, 15(1), jan 2016.

[38] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022.

[39] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing Billion-Scale model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564. USENIX Association, July 2021.

[40] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, Fengwei Zhang, and Heming Cui. SOTER: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 723–738, Carlsbad, CA, July 2022. USENIX Association.

[41] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. Slora: Scalable serving of thousands of lora adapters. *Proceedings of Machine Learning and Systems*, 6:296–311, 2024.

[42] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[43] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[44] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv preprint arXiv:2312.12456*, 2023.

[45] Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. Turbo sparse: Achieving llm sota performance with minimal activated parameters. *arXiv preprint arXiv:2406.05955*, 2024.

[46] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[47] ShareGPT Team. https://sharegpt.com, 2023.

[48] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[50] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681–696, Carlsbad, CA, October 2018. USENIX Association.

[51] Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*, 2024.

[52] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[53] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

[54] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, and Dan Meng. Enabling rack-scale confidential computing using heterogeneous trusted execution environment. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1450–1465, 2020.

[55] Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. Understanding the security of discrete gpus. In *Proceedings of the General Purpose GPUs*, GPGPU-10, page 1–11, New York, NY, USA, 2017. Association for Computing Machinery.