Cost-Efficient LLM Training with Lifetime-Aware Tensor Offloading via GPUDirect Storage

Ziqi Yuan[1], **Haoyang Zhang**[1], **Yirui Eric Zhou**[1],
Apoorve Mohan[2], **I-Hsin Chung**[2], **Seetharami Seelam**[2], **Jian Huang**[1]
[1]University of Illinois Urbana-Champaign, [2]IBM Research
{ziqiy6, zhang402, yiruiz2, jianh}@illinois.edu
apoorve.mohan@ibm.com, {ihchung, sseelam}@us.ibm.com

Abstract

We present the design and implementation of a new lifetime-aware tensor offloading framework for GPU memory expansion using low-cost PCIe-based solid-state drives (SSDs). Our framework, TERAIO, is developed explicitly for large language model (LLM) training with multiple GPUs and multiple SSDs. Its design is driven by our observation that the active tensors take only a small fraction (1.7% on average) of allocated GPU memory in each LLM training iteration, the inactive tensors are usually large and will not be used for a long period of time, creating ample opportunities for offloading/prefetching tensors to/from slow SSDs without stalling the GPU training process. TERAIO accurately estimates the lifetime (active period of time in GPU memory) of each tensor with the profiling of the first few iterations in the training process. With the tensor lifetime analysis, TERAIO will generate an optimized tensor offloading/prefetching plan and integrate it into the compiled LLM program via PyTorch. TERAIO has a runtime tensor migration engine to execute the offloading/prefetching plan via GPUDirect storage, which allows direct tensor migration between GPUs and SSDs for alleviating the CPU bottleneck and maximizing the SSD bandwidth utilization. In comparison with state-of-the-art studies such as ZeRO-Offload and ZeRO-Infinity, we show that TERAIO improves the training performance of various LLMs by 1.47× on average, and achieves 80.7% of the ideal performance assuming unlimited GPU memory.

1 Introduction

Large language models (LLMs) have been widely employed in various application domains [31, 46, 42]. However, training LLMs is still a well-known challenging problem, as its memory demand is increasing at a much faster speed than the scaling speed of GPU memory [18, 25]. Scaling the GPU memory capacity is both technically challenging and prohibitively expensive. Due to space constraints and DRAM scaling issues [19, 15], it is hard to scale up the GPU memory capacity on each server machine. Scaling out LLM training with a cluster of GPU servers can increase the aggregated memory capacity, however, it inevitably increases operational cost, placing a major barrier for researchers and developers to entry into cutting-edge LLM development.

To overcome the GPU memory wall, prior studies proposed expanding GPU memory with external memory devices. One common approach is to allow tensor offloading from GPU memory to host memory [17, 39, 11, 44, 32, 41, 43, 20]. However, due to the fundamental DRAM scalability challenge, such an approach is still limited by the host memory. Recent studies have extended tensor offloading to PCIe-based SSDs that offer larger capacity at a much lower cost [1, 37, 16, 23, 47, 51, 29, 50]. But due to the incapability of efficiently utilizing the SSD bandwidth and hiding the slow SSD accesses, these offloading solutions still deliver suboptimal performance. For instance, ZeRO-infinity [37] enabled the offloading of tensors to SSDs at the granularity of deep neural network

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

 https://doi.org/10.52202/085713-1653

(DNN) layers, its coarse-grained offloading/prefetching scheme wastes not only the limited SSD bandwidth but also the precious GPU memory space, leaving the solution less attractive in practice.

Ideally, we wish to expand GPU memory with low-cost SSDs while achieving similar training performance as the ideal case assuming GPUs have unlimited on-board memory. To this end, we present a new tensor offloading framework – TERAIO, which enables fine-grained offloading of tensors in an accurate fashion based on their activity patterns in GPUs, for best utilizing both SSD bandwidth and GPU memory. Our study of tensor activity patterns (§2) in LLM training shows (1) the active tensors, which are used in the current kernel during LLM training, consume only a small portion (1.7% on average) of requested GPU memory in total; (2) many inactive tensors are large and occupy a substantial GPU memory space in each training iteration; but (3) these inactive tensors are not used in the training for a long period of time, depending on the computation intensity of LLMs.

With these insights, TERAIO develops a lifetime-aware tensor offloading mechanism following three design principles: (1) offloading large inactive tensors to SSDs can save precious GPU memory and maximize SSD bandwidth utilization during the training process; (2) the distribution of their inactive periods of time will help TERAIO decide which inactive tensor should be offloaded at what time, and similarly, which tensor should be prefetched at what time; (3) precisely scheduling tensor offloading and prefetching in consideration of the available SSD bandwidth will help TERAIO effectively overlap the tensor movement with GPU computation. Our roofline model analysis (§2.2) shows that, given each GPU connected to multiple commodity SSDs today, the aggregated storage I/O bandwidth is sufficient to meet the tensor migration requirement without hurting the GPU training process.

To fulfill the design principles discussed above, we develop TERAIO with three major components: (1) a tensor lifetime profiler that can extract tensor activity patterns (e.g., tensor size and lifetime) in advance with the assistance of deep learning compilers such as PyTorch, (2) a lifetime-aware tensor migration algorithm that can generate optimal tensor offloading/prefetching plans based on the learned tensor activity patterns, and (3) a tensor migration engine that will execute the generated offloading/prefetching plans with efficient direct data transfer between GPUs, host memory, and SSDs. We present each of these components as follows.

An open-source tensor lifetime profiler. TERAIO conducts the profiling of the tensor size and lifetime distributions by running the first few iterations of LLM training on the target GPU setting. As the computation and dataflow patterns of each iteration are almost the same, the profiling results can accurately represent the generic patterns of the entire LLM training process. To track the metadata information of each tensor, we instrument the automatic operator generator in PyTorch rather than intrusively instrument the source code of each generated operator. Therefore, the proifler requires minimal code modifications to PyTorch. As the execution of LLM on GPUs has highly predictable dataflow patterns, TERAIO uses the execution time of GPU kernels to accurately estimate the tensor lifetime (i.e., the length of the active and inactive period of time). With the knowledge of tensor activity patterns, TERAIO will create the tensor offloading/prefetching plans in advance.

Lifetime-aware tensor migration algorithm. TERAIO prioritizes offloading large tensors with long inactive period of time to SSDs, for fully utilizing the available storage I/O bandwidth. For tensors that have short inactive periods of time, TERAIO will make the best effort to retain them in GPU memory, for avoiding unnecessary migration overhead. As host memory and SSD offer different capacities, bandwidths, and costs, TERAIO prefers to offload tensors to SSDs for taking advantage of their large capacity and low cost. However, when the SSD bandwidth is saturated at runtime, TERAIO will use the host memory as the offloading destination. Given an execution plan for each LLM training iteration, TERAIO will iteratively search for the best offloading candidate based on the tensor size and lifetime, until the required GPU memory is below the capacity limit. After that, TERAIO will generate an optimized tensor migration plan by adding corresponding offloading and prefetching instructions in the compiled LLM training program.

Tensor migration engine using GPUDirect storage. Following the tensor migration plan integrated into the compiled LLM training program, the tenor migration engine of TERAIO will offload/prefetch tensors to/from SSDs or host memory at runtime. For the tensor migration between GPU memory and SSD, TERAIO uses GPUDirect storage to enable direct data transfer between GPUs and SSDs, therefore, it can bypass the host CPU to alleviate the scalability bottleneck and maximize SSD bandwidth utilization. When the available SSD bandwidth is insufficient to support tensor offloading and prefetching, TERAIO will migrate tensors to the host memory. To track the latest locations of

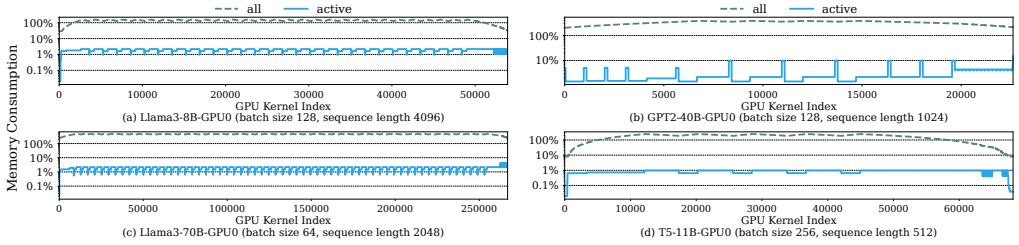
tensors (GPU memory, host memory, or SSDs), TERAIO indexes tensors with their identification numbers using hash maps.

We implement the core components of TERAIO based on PyTorch. Therefore, TERAIO does not require any code modifications to LLM training programs. To evaluate the efficiency of TERAIO, we train a set of Llama and Granite models with different batch sizes and sequence lengths using TorchTitan [22] on a GPU server that has two NVIDIA H100 GPUs and eight PCIe-based SSDs. In comparison with state-of-the-art offloading solutions ZeRO-Offload [40] and ZeRO-Infinity [37], TERAIO improves the training performance by 1.47× on average, achieves 80.7% of the ideal performance assuming unlimited GPU on-board memory, and delivers 1.45× improvement on cost efficiency for LLM training. In summary, we make the following contributions.

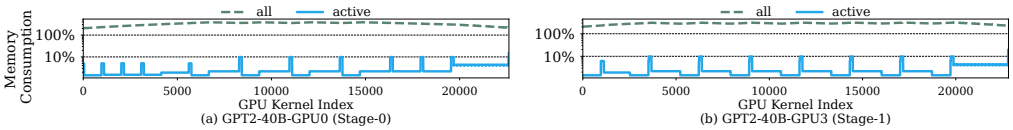
- We conduct a quantitative characterization study of tensor memory usage when training different LLMs on multiple GPUs, and show that the high compute intensity of modern LLMs provide rich opportunities for tensor offloading.
- We develop a lightweight tensor lifetime profiler based on PyTorch, which can learn tensor activity patterns for multi-GPU LLM training.
- We design a lifetime-aware tensor migration planning algorithm that optimizes offloading/prefetching decisions based on tensor activity patterns, GPU memory capacity, and the available migration bandwidth.
- We implement a transparent tensor migration engine that enables direct data transfer between GPU and SSDs, alleviating the scalability bottleneck on the host.
- We conduct a thorough evaluation of TERAIO with the training of various LLMs, demonstrating significant improvement on training performance and cost efficiency, compared to state-of-the-art offloading solutions.

2 Characterization Study of Tensor Activity Patterns in LLM Training

In this section, we present our characterization study of tensor activity patterns in LLM training. To facilitate our study, we utilize our tensor lifetime profiler, which will be discussed in §3.1, to analyze the distributions of tensor sizes and lifetimes during the LLM training. We use two NVIDIA H100 GPUs and 2-stage `1f1b` pipeline parallelism [28] in our experiments. Our study covers a variety of LLMs with different architectures, including decoder-only models such as Llama3-8B (batch size of 128, sequence length of 4096), Llama3-70B (batch size of 256, sequence length of 2048) [4], and GPT2-40B (batch size of 16, sequence length of 1024) [35], as well as encoder-decoder models like T5-11B (batch size of 64, sequence length of 512) [7]. For models that require more memory than GPU memory capacity, we offload tensors not needed by the current kernel to SSDs. We summarize our study results as follows.



(a) Memory consumption of different LLMs on each GPU.



(b) Memory consumption of different stages in the pipeline parallelism.

Figure 1: Memory consumption of all and active tensors (w.r.t. the GPU memory capacity) in one iteration of the parallel training program. Logarithmic scale is used in the presentation.

https://doi.org/10.52202/085713-1653

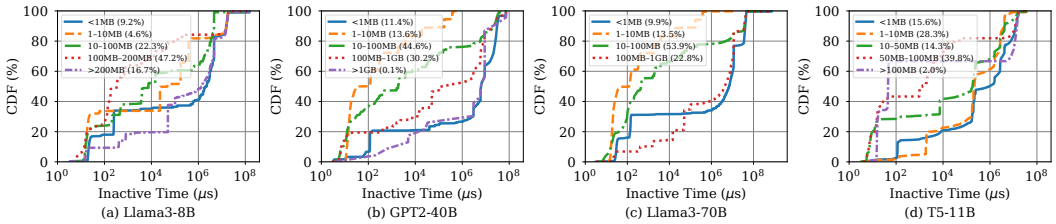


Figure 2: The distribution of inactive periods of tensors.

2.1 Rich Opportunities for Tensor Migration

Small memory requirement of active tensors. We first study the memory demand and usage of tensors in each training iteration. We define the tensors that are currently used by a running GPU kernel as ***active tensors***. We present the memory consumption of active tensors used by each GPU kernel in Figure 1. Figure 1 (a) shows the memory consumption of tensors in different models, Figure 1 (b) shows the memory consumption of tensors across different pipeline stages. For all LLMs examined in our study, the active tensors account for only less than 14% (1.7% on average) of the total GPU memory capacity, although their total memory usage greatly exceeds the GPU memory capacity. Most tensors in GPU memory are inactive and can be offloaded to low-cost SSDs, thus, we can best utilize the GPU memory for tensors that will be used by kernels in the near future.

Long inactive periods of inactive tensors. To understand how long the inactive tensors remain inactive and how much GPU memory they consume, we study the distribution of their inactive periods, as shown in Figure 2. For all the models we study, most tensors have sizes that range from 10MB to 1GB. We observe that more than 40% of these tensors remain inactive for more than 10^4 microseconds. These inactive periods are longer than the time needed to migrate these tensors to SSDs at a bandwidth of 6.5 GB/s. With this insight, we can ensure that these tensors can be migrated efficiently without introducing negative impact on the training performance.

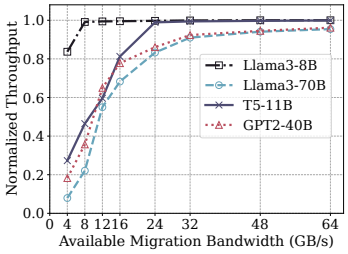


Figure 3: Roofline analysis with different migration bandwidths. The training performance is normalized to the ideal case assuming GPU memory is infinite.

The long inactive periods are the cause of the sparse tensor access pattern and high compute intensity of LLMs. From a spatial perspective, although LLMs have tens or hundreds of layers, many tensors are used within only a single layer. From a temporal perspective, the compute-intensive kernels (e.g., attention) in each layer take a considerable amount of time, providing rich opportunities for TERAIO to overlap the computation with the migration of inactive tensors.

We also observe that, compared with traditional DNN models, the higher compute intensity and larger model sizes of LLMs lead to substantially longer inactive periods. For example, in BERT-Large [3], 48% of tensors are larger than 100MB, and the inactive periods of more than 60% of these large tensors are two orders of magnitude shorter than those in LLMs.

2.2 Bandwidth Requirement for Tensor Migration

We now study how much migration bandwidth is needed for offloading to achieve near-ideal training performance. We quantify the roofline performance of different LLM models under different migration bandwidths available to each GPU. To facilitate this study, we build a performance model to estimate the training time. In the model, we assume that each kernel's execution time is the same as the value collected in our characterization study. We simulate tensor migration at the designated bandwidth, and check whether the tensors needed by the kernel are already in GPU memory or not. If they are still being migrated due to limited SSD bandwidth, the waiting time for the migration is added to the total training time. Figure 3 shows the normalized roofline training throughput of LLMs under different migration bandwidths. We observe that a bandwidth of 32 to 48 GB/s is sufficient to achieve near-ideal performance for LLMs. Such a bandwidth requirement can be easily achieved by aggregating multiple commodity SSDs (e.g., an SSD array), demonstrating the feasibility of TERAIO.

https://doi.org/10.52202/085713-1653

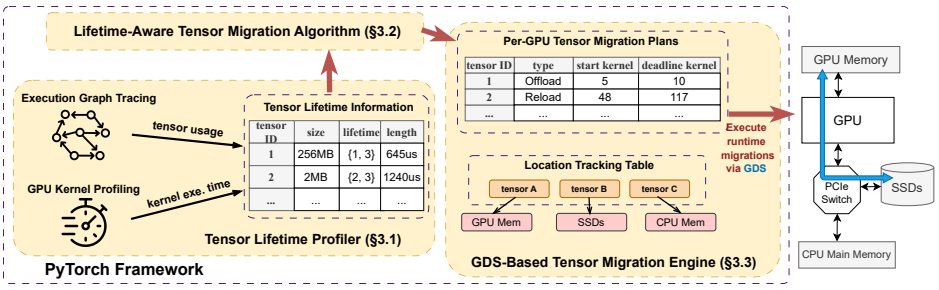


Figure 4: System overview of TERAIO.

3 TERAIO Design and Implementation

We show the system overview of TERAIO in Figure 4. Given an LLM, TERAIO's tensor lifetime profiler works with PyTorch to track tensor sizes and lifetimes (§3.1). In the first few training iterations, TERAIO traces the execution graph and collects the execution time of each kernel. Since the training follows the same execution graph in subsequent iterations, the tensor activity patterns remain the same. The tensor migration algorithm (§3.2) creates a tensor migration plan that (1) maximally overlaps computation and migration, and (2) minimizes migration traffic. The algorithm iteratively selects the best offloading candidates until the required GPU memory fits within the actual GPU memory capacity. For the migration destination, it prefers to migrate tensors to SSDs. Once the SSD bandwidth is saturated, it also uses available CPU memory. During LLM training, TERAIO's tensor migration engine transparently executes the migration plan (§3.3).

3.1 Tensor Lifetime Profiler

Tracking tensors. TERAIO instruments PyTorch framework to track tensors and measure kernel execution time at runtime. A tensor is considered active in one of the following three scenarios.

First, a tensor is active when it is the input or output of a PyTorch CUDA operator. However, instrumenting PyTorch to track every operator is challenging, as there are thousands of operators. Instead, we leverage PyTorch's automatic operator generator, which produces source code for each operator, to insert profiling code that will mark all input and output tensors as active when the operator is executed at runtime. Second, for tensors that are involved in inter-GPU communication, they should be active in GPU memory. Third, a tensor is considered active when PyTorch explicitly checks whether it resides in GPU memory. This happens when updating optimizer states. For the second and third scenarios, since there are only a few communication operators and PyTorch checks in total, we directly set the corresponding tensors as active in the source code.

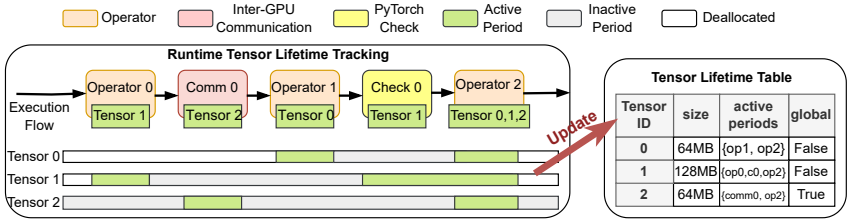
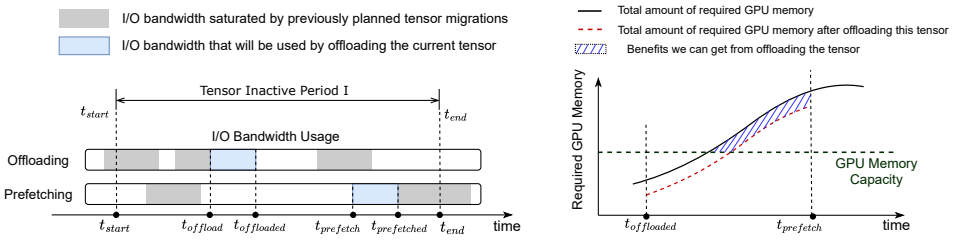


Figure 5: TERAIO tracks and analyzes tensor activity patterns for the tensor migration algorithm.

Analyzing tensor activity patterns. Figure 5 shows how tensor information is collected at runtime. To understand when a tensor consumes GPU memory and when it must reside in GPU memory, we need to collect its *tensor size and active time*. When an operator is executed, the instrumented code records the *tensor size* and *active time* for the corresponding tensor. The profiler will calculate the inactive time period based on the duration between its active states. Specifically, for intermediate tensors such as gradients and activations (Tensor 0 and Tensor 1 in Figure 5) that will be deallocated immediately after its computation completes, we quantify its inactive time period as the time interval between the two active periods. For global tensors such as model weights and optimizer states (Tensor 2 in Figure 5) that are used across multiple training iterations, they are allocated before training starts and never deallocated during training. Therefore, for some cases, the profiler may need to calculate its inactive period based on the active states across two iterations.

https://doi.org/10.52202/085713-1653



(a). I/O bandwidth usage affects actual migration time (b). The benefit of offloading one inactive tensor

Figure 6: Illustration examples to explain the key insights of lifetime-aware migration algorithm.

3.2 Lifetime-aware Tensor Migration Algorithm

The lifetime-aware tensor migration algorithm iteratively finds the best offloading candidates in each LLM training iteration, until the required GPU memory is below the capacity limit. By tracking the amount of required GPU memory and the storage I/O bandwidth utilization, the algorithm is able to evaluate the potential benefits of tensor offloading. We discuss our key ideas as follows.

Storage I/O bandwidth-aware planning. To offload an inactive tensor, we wish to keep it out of GPU memory as long as possible. Therefore, ideally, we would offload it as soon as it becomes inactive and prefetch it instantly before it is needed by the subsequent kernel. However, in reality, the time when we can offload and prefetch the inactive tensor is greatly affected by the storage bandwidth usage. For example, in Figure 6(a), the inactive period *I* of the tensor is from t_{start} to t_{end}. However, since the I/O bandwidth is occupied by previously planned migrations, we have to delay the actual offloading until $t_{offload}$ and start the actual prefetching by $t_{prefetch}$. This means that we can only reduce the memory consumption from $t_{offloaded}$ to $t_{prefetch}$. When planning tensor migrations, our algorithm tracks the estimated storage I/O bandwidth usage and calculates the reduction in memory consumption in an I/O-aware manner (Line 4-5 and 16-17 in Algorithm 1).

Algorithm 1 Lifetime-Aware Tensor Migration Planning

Require: Set of tensor inactive periods $I = \{(t_i, s_i, start_i, end_i)\}$ where t_i is tensor ID, s_i is size, and $[start_i, end_i]$ defines the kernel range of inactivity; GPU memory capacity M_{GPU}; Estimation of total amount of required GPU memory $M = [m_0, m_1, ..., m_{N-1}]$ across N kernels; Kernel execution times $T = [\tau_0, \tau_1, ..., \tau_{N-1}]$; I/O bandwidth usage states
Ensure: Migration plan list $P = \{(t_i, trigger\_time, deadline, target)\}$
1: Initialize migration plan list $P = \{\}$ and estimation of required GPU memory $M' = M$
2: **while** peak memory consumption $\max(M') > M_{GPU}$ **do**
3: **for** each inactive period $(t, s, start, end) \in I$ **do**
4: Determine earliest feasible offload completion time $t_{offloaded}$ by analyzing available offloading bandwidth
5: Determine latest required prefetch start time $t_{prefetch}$ by analyzing available prefetching bandwidth
6: **if** $t_{offloaded} < t_{prefetch}$ **then**
7: Identify critical memory pressure regions $C = \{k | M'[k] > M_{GPU}, k \in [t_{offloaded}, t_{prefetch}]\}$
8: Calculate the offloading benefit $B = s \times \sum_{k \in C} \tau_k$
9: **end if**
10: **end for**
11: Select inactive period with best offloading cost-benefit $(B/s)_{max}$
12: **if** no viable offloading candidate exists **then**
13: **break**
14: **end if**
15: Add migration plans $(t, start, t_{offloaded}, SSD/CPU)$ and $(t, t_{prefetch}, t_{prefetched}, GPU)$ to P
16: Update M' by subtracting tensor size s from affected kernels
17: Update I/O bandwidth usage states
18: Remove selected inactive period from consideration
19: **end while**
20: **return** P

Quantify the benefit and cost of tensor offloading. To fully utilize the available I/O bandwidth, we want to prioritize offloading large tensors with long inactive periods. Following this principle, our algorithm searches for the best offloading candidate by estimating its benefit and cost. To quantify the benefit, at a given time *T*, we define *critical memory pressure* as the part of GPU memory consumption that exceeds the capacity. The benefit of a tensor migration is defined as the integral of the reduction in critical memory pressure over time, as illustrated by the shaded area in Figure 6(b). We quantify the cost as the sum of offloading and prefetching time of tensors. TERAIO's migration algorithm sorts all candidates by their benefit-to-cost ratio. It then selects the tensor with the highest ratio for migration in the current iteration. This procedure is shown in Line 6-11 in Algorithm 1.

https://doi.org/10.52202/085713-1653

Decide offloading destination. TERAIO prioritizes SSDs because of their large capacity and low cost. When the estimated SSD bandwidth is saturated, TERAIO will make the best effort to migrate tensors to the host memory. TERAIO allows users to define the maximum amount of host memory that can be used for tensor migration. Internally, the migration algorithm tracks the estimated host memory consumption. If it has already reached the user-defined limit, even if SSD bandwidth is saturated, TERAIO will not offload tensors to the host memory.

Minimize kernel stalls. If the required GPU memory exceeds the available GPU memory capacity, and TERAIO cannot offload tensors to SSDs or host memory, TERAIO will stall kernel execution to wait for more inactive tensors to be offloaded. It will also wait for the tensors needed by the kernel to be migrated back into GPU memory. This would cause storage I/O bandwidth contention. Since the next kernel will stall until the needed tensors are migrated to GPU memory, these migrations should complete as early as possible to avoid stalling the GPU training process. Therefore, TERAIO marks these migrations that must finish before the next kernel as 'urgent'. At runtime, the tensor migration engine (§3.3) always prioritizes these urgent migrations over other pending migrations.

Compilation cost. The overhead of generating tensor migration plans based on the lifetime analysis is shown in Table 1. Although this compilation step introduces additional latency, its cost is less significant in practice because it enables efficient tensor offloading, which reduces overall model training time. After the plan generation, we do not need to recompile ML models, as the corresponding offloading and prefetching instructions have been integrated into the execution graph. The details are described in Section 3.3.

Table 1: Compilation time for different models.

Model	Time(s)
Llama3-8B	31.2
Granite-code-base-8B	37.9
Llama3-70B	396.6

3.3 Tensor Migration Engine Using GPUDirect Storage

To execute the migration plan, we need to locate the addresses of the tensors to be migrated, based on the tensor identifier. TERAIO maintains a hashmap-based tensor location table in PyTorch to map tensor identifiers to their current devices and addresses.

The migration engine migrates tensors between GPU memory and external memory. For tensor migrations between GPU memory and SSDs, we use GPUDirect storage to achieve direct data transfer between them. Our choice of GPUDirect storage is motivated by the potential scalability limitations of host CPU in multi-GPU systems. For example, on an 8-GPU system, to achieve near-ideal performance, each GPU requires 32 to 48 GB/s bidirectional bandwidth (see §2). With GPUDirect storage, we can directly connect 8 SSDs to each GPU via PCIe Gen5 switches to meet the bandwidth demand. For the conventional approach that uses the host CPU to first read data from SSDs to the host memory, and then uses `cudaMemcpy` to move data to the GPU, the redundant data copy not only causes performance overhead but also wastes precious CPU cycles [8, 33]. But as discussed, TERAIO still supports migration between GPUs and CPU memory.

4 Experiments

We show that (1) TERAIO outperforms state-of-the-art offloading frameworks by 1.47× on average when training LLMs that greatly exceed GPU memory capacity (§4.2); (2) Compared to the case of training LLMs using only GPU memory, TERAIO reduces the cost by up to 5.41× (§4.3); (3) Compared to existing offloading frameworks, TERAIO improves the cost efficiency by 1.45× on average (§4.3); (4) TERAIO achieves better throughput than ZeRO-Infinity even with less CPU memory and fewer SSDs (§4.4).

4.1 Experimental Setup

Models. We evaluate TERAIO with Llama3-8B, Llama3-70B [4], and Granite-code-base-8B [13]. We use C4 [36] as our training dataset. To study how different memory demands impact the performance of TERAIO, we use batch sizes ranging from 16 to 128 and sequence lengths from 1,024 to 8,192.

Hardware configuration and ML framework. Table 2 shows the hardware configuration used in our experiments. Due to the limited PCIe slots on our machine, we can only install at most 8 PCIe SSDs. When evaluating TERAIO, we use 2 H100 GPUs and 2 RAID-0 arrays with 4 SSDs in each array. Each RAID-0 array is logically assigned to one GPU, providing approximately 16 GB/s bandwidth for tensor migrations. We use PyTorch 2.5.0 [30] and TorchTitan [22] to train LLMs.

 https://doi.org/10.52202/085713-1653

Baselines. We compare TERAIO with the *Ideal case, ZeRO-Offload, and ZeRO-Infinity*. The ***Ideal case*** assumes that all GPUs in the system have infinite on-board memory, which gives the theoretical best training performance. ***ZeRO-Offload*** [39] and ***ZeRO-Infinity*** [37] are popular offloading-based training systems. ZeRO-Offload offloads tensors from GPU memory only to CPU memory, while ZeRO-Infinity leverages both SSDs and CPU memory to expand GPU memory. **TERAIO-*SSD*** and **TERAIO-*Mixed*** are two variations of TERAIO. TERAIO-SSD only migrates tensors to low-cost SSDs, while TERAIO-Mixed uses both SSDs and CPU memory. To make a fair comparison, we let TERAIO-Mixed use the same amount of CPU memory for tensor migration as ZeRO-Infinity. For ZeRO, we preserve most of its default parameters and fine-tune several parameters to maximize its throughput. We list its detailed configurations in Table 4 (Appendix B).

Table 2: Our GPU server configuration.

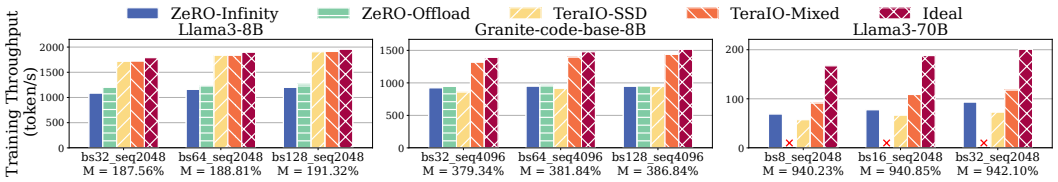
GPU	2× NVIDIA H100 NVL
GPU Memory	94GB HBM per GPU
CPU	2× AMD EPYC 9334
CPU Memory	1.5TB DDR5 (64GB ×24)
Interconnect	PCIe Gen5
SSDs	8× Samsung 990 PRO 2TB
SSD Read/Write Bandwidth	6.7/6.5 GB/s per SSD

In our evaluation, we aim to compare TERAIO with the best performance achievable by ZeRO. Therefore, for the parallelization strategy, we use tensor parallelism for ZeRO series, as it delivers optimal multi-GPU training performance. Moreover, we split each batch into multiple micro-batches in order to amortize the well-known performance bottleneck [16] of ZeRO's CPU-based optimizers. In addition, although activation checkpointing can reduce memory consumption, we disable it because it degrades ZeRO's training throughput.

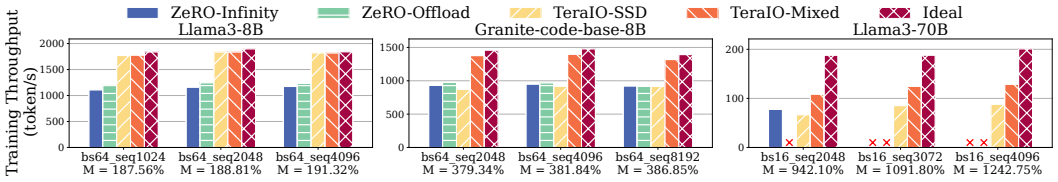
In terms of training precision, we use full-precision training in all experiments. Though mixed-precision training [26] is popular, the different memory requirements of the mixed-precision training strategies used by TorchTitan and ZeRO will lead to an unfair comparison. Specifically, when we enable mixed-precision training, in the ZeRO series, all tensors in the GPU are represented by 16-bit floating points, while in TorchTitan, most tensors, including model weights, gradients, and optimizer states still remain in 32-bit floating point format. Such differences in numerical formats of tensors can lead to significant differences in GPU memory requirements for the same model, resulting in an unfair scenario where TERAIO has to migrate larger tensors than ZeRO.

4.2 End-to-end Performance

We show the end-to-end average training throughput of Llama3-8B, Granite-code-base-8B and Llama3-70B with different batch sizes and sequence lengths in Figure 7. On average, TERAIO outperforms ZeRO-Offload and ZeRO-Infinity by 1.47×. Compared to the ideal system assuming unlimited GPU memory, TERAIO achieves 80.7% of the ideal performance.



(1). Average training throughput with different batch sizes.



(2). Average training throughput with different sequence lengths.

Figure 7: Average training throughput of Llama3-8B, Granite-code-base-8B and Llama3-70B with different batch sizes and sequence lengths. `bs` is the batch size. `seq` is the sequence length. `M` is the overall peak memory consumption of the LLM training on one GPU w.r.t. the GPU memory capacity. "×" means the framework failed to train this model due to out-of-memory errors.

Training throughput. As shown in Figure 7, when training Llama3-8B and Granite-8B, ZeRO-Offload achieves 65.9% and 65.5% of the ideal performance, respectively. Though ZeRO-Infinity takes extra time to migrate tensors further from CPU memory to SSDs, its training throughput

https://doi.org/10.52202/085713-1653

is slightly lower than ZeRO-Offload. Such a subtle performance difference comes from the high aggregate bandwidth of 8 SSDs and the relatively small sizes of optimizers and parameters offloaded to SSDs. For Llama-70B, ZeRO-Offload fails because the host memory capacity is too small to store offloaded tensors of such a large model. Since SSDs offer larger capacity, ZeRO-Infinity can still train Llama3-70B. However, it only achieves 43.0% of the ideal performance, because its coarse-grained offloading scheme cannot efficiently utilize the limited SSD bandwidth to migrate larger tensors.

TERAIO outperforms the ZeRO series by up to 1.59×. For Llama3-8B, both TERAIO-Mixed and TERAIO-SSD can achieve near-ideal performance, demonstrating the effectiveness of our lifetime-aware tensor migration algorithm in choosing the most beneficial tensor to migrate. Moreover, we find that even though 2 H100 GPUs can provide sufficient GPU memory to train Llama3-8B without memory expansion, TERAIO allows us to increase the (micro-)batch size, improving the throughput by 9%. For Granite, TERAIO-SSD achieves similar performance to the ZeRO series, while TERAIO-Mixed can still deliver near-ideal performance by utilizing CPU memory. The result shows that as the model size increases, achieving near-ideal performance requires not only our lifetime-aware tensor migration algorithm, but also migration bandwidth higher than what 4 SSDs can provide. For Llama-70B, since its memory requirement significantly exceeds GPU memory capacity, higher migration bandwidth is needed. Even TERAIO-Mixed can only achieve 59.7% of the ideal performance. Nonetheless, it still outperforms ZeRO-Infinity by 1.33×.

Impact of varying batch size and sequence length. As batch size and sequence length vary, the training throughput of TERAIO-Mixed is always the closest to the ideal throughput among all offloading frameworks, while TERAIO-SSD delivers similar or better performance than the ZeRO series. For the same model, increasing the batch size doesn't raise memory requirements, since each batch is split into micro-batches that contain the same number of training samples. However, when the sequence length increases, the memory requirement also increases. Therefore, when the sequence length is longer than 3,072, ZeRO-Infinity fails to train the model since the GPU memory capacity is smaller than the activation tensors of the model. In contrast, TERAIO can train all model configurations because it can offload any inactive tensor to save memory.

Breakdown of training time. To better understand the source of TERAIO's performance gains, we also show the breakdown of the end-to-end training time in Figure 8. It includes three components: (a) the time when tensor migrations stall GPU computation, (b) the time when tensor migrations overlap with GPU computation, and (c) the remaining GPU computation time.

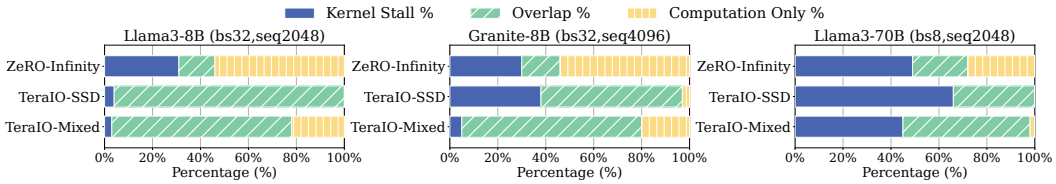


Figure 8: Latency breakdown of training iterations.

As shown in Figure 8, compared to ZeRO-Infinity, TERAIO-Mixed has less stall time, as it maximizes the overlap of tensor migrations with computation. TERAIO-Mixed outperforms ZeRO-Infinity for the design differences in offloading granularity and migration strategy. As for offloading granularity, ZeRO-Infinity selects the offloading of tensors at the layer granularity of ML models. This introduces burst I/O patterns and can underutilize migration bandwidth. In contrast, TERAIO selects tensor offloading at the GPU kernel granularity based on our precise tensor lifetime analysis. This approach ensures high and consistent migration bandwidth utilization, as shown in Figure 10 (Appendix C). As for migration strategy, ZeRO-Infinity uses a heuristic-based policy to decide which tensors should be offloaded and when the offloading should start, regardless of the storage bandwidth usage. It lacks global optimization and produces unpredictable I/O patterns. In contrast, TeraIO quantifies the benefit and cost of tensor offloading via I/O-aware planning (Section 3.2), which generates globally optimized migration plans that maximize I/O bandwidth utilization.

https://doi.org/10.52202/085713-1653

4.3 Training Cost

To evaluate the cost of TERAIO, we summarize the prices of different devices used in each baseline setup in Table 3. TERAIO-SSD needs a server with only 128GB of CPU memory and 8 SSDs since it migrates tensors only to SSDs, while TERAIO-Mixed uses both SSDs and 1TB CPU memory for more efficient tensor migration. Since ZeRO-Offload and ZeRO-Infinity consume a large amount of CPU memory to train LLMs, they both need a server with 1TB of memory. ZeRO-Infinity additionally uses 8 SSDs in our evaluation. We also compare with the PureGPU setup, in which all tensors are kept within GPU memory. To provide enough GPU memory to train Llama3-70B in this setup, we need to pay $499,591.40 for two servers, each equipped with 8 H100 GPUs. In comparison, TERAIO-SSD and TERAIO-Mixed save costs by 5.88× and 5.41×, respectively. Compared to ZeRO-Offload and ZeRO-Infinity, since we have similar machine setups, TERAIO's 1.47× training performance improvement translates into 1.45× and 1.47× improved cost efficiency, respectively.

Table 3: Cost of each device used by TERAIO and other baselines for Llama3-70B training. Prices are quoted from *Exxact* [5]. The PureGPUs setup contains the minimum number of 8-GPU H100 servers capable for training Llama3-70B without offloading tensors to host memory or SSDs.

	Server (with 2 H100 GPUs) with 128GB memory	Server (with 2 H100 GPUs) with 1TB memory	2× Server (with 8 H100 GPUs) with 128GB memory	8× Samsung 990PRO SSD 2TB
Cost ($)	84,139.9	91,047.9	499,591.4	1,360
TeraIO-SSD	✓			✓
TeraIO-Mixed		✓		✓
ZeRO-Offload		✓		
ZeRO-Infinity		✓		✓
PureGPUs			✓	

4.4 Impact of Varying Number of SSDs and CPU Memory Capacity

Figure 9 shows the training throughput of TERAIO as we vary the available CPU memory capacity and the number of SSDs used for each GPU. The performance of TERAIO scales favorably as we increase the number of SSDs or the CPU memory capacity. For Llama-8B, TERAIO achieves near ideal performance with only 2 SSDs and 64 GB of CPU memory. Even for Llama-70B, we observe that using only 2 SSDs and 512GB of CPU memory can still achieve 73.1% of the training throughput obtained with 4 SSDs and 1,024GB of CPU memory. These results validate our LLM characterization study's key observation: with *lifetime-aware* tensor offloading, we can achieve good performance even with limited hardware resources.

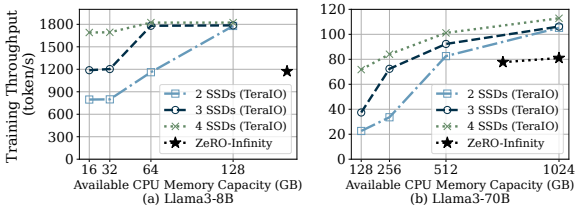


Figure 9: Training throughput as we vary the CPU memory capacity and the number of SSDs per GPU.

Compared to ZeRO-Infinity, TERAIO significantly reduces the CPU memory capacity requirement while achieving better performance. ZeRO-Infinity requires 170GB and 770GB of CPU memory to train Llama-8B and Llama-70B, respectively, as it has to offload gradients and optimizers into CPU. In comparison, TERAIO achieves better performance even with more limited hardware resources, as it efficiently utilizes both the large capacity of SSDs and extra bandwidth of CPU memory.

5 Conclusion

We present TERAIO, a lifetime-aware tensor offloading framework that can accurately plan and execute fine-grained tensor offloading and prefetching instructions for LLM training. With predictable tensor activity patterns, TERAIO best utilizes precious GPU memory for accelerating GPU training process, while leveraging large-capacity SSDs for lowering training cost. Compared to existing tensor offloading work, TERAIO provides a more practical and cost-efficient solution for LLM training.

Acknowledgment

We thank the anonymous reviewers for their insightful feedback. We thank the members in the Systems Platform Research Group (Illinois PlatformX) at UIUC for constructive discussions. This work was partially supported by the Hybrid Cloud and AI program at the IBM-Illinois Discovery Accelerator Institute (IIDAI), and NSF under the grants CAREER CNS-2144796 and CCF-2107470.

References

[1] Jonghyun Bae, Jongsung Lee, Yunho Jin, Sam Son, Shine Kim, Hakbeom Jang, Tae Jun Ham, and Jae W Lee. {FlashNeuron}:{SSD-Enabled}{Large-Batch} training of very deep neural networks. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 387–401, 2021.

[2] Olivier Beaumont, Lionel Eyraud-Dubois, and Alena Shilova. Efficient combination of rematerialization and offloading for training dnns. *Advances in Neural Information Processing Systems*, 34:23844–23857, 2021.

[3] 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.

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

[5] EXXACT. Exxact. `https://www.exxactcorp.com/`, 2025.

[6] Talia Gershon, Seetharami Seelam, Brian Belgodere, Milton Bonilla, Lan Hoang, Danny Barnett, I-Hsin Chung, Apoorve Mohan, Ming-Hung Chen, Lixiang Luo, Robert Walkup, Constantinos Evangelinos, Shweta Salaria, Marc Dombrowa, Yoonho Park, Apo Kayi, Liran Schour, Alim Alim, Ali Sydney, Pavlos Maniotis, Laurent Schares, Bernard Metzler, Bengi Karacali-Akyamac, Sophia Wen, Tatsuhiro Chiba, Sunyanan Choochotkaew, Takeshi Yoshimura, Claudia Misale, Tonia Elengikal, Kevin O Connor, Zhuoran Liu, Richard Molina, Lars Schneidenbach, James Caden, Christopher Laibinis, Carlos Fonseca, Vasily Tarasov, Swaminathan Sundararaman, Frank Schmuck, Scott Guthridge, Jeremy Cohn, Marc Eshel, Paul Muench, Runyu Liu, William Pointer, Drew Wyskida, Bob Krull, Ray Rose, Brent Wolfe, William Cornejo, John Walter, Colm Malone, Clifford Perucci, Frank Franco, Nigel Hinds, Bob Calio, Pavel Druyan, Robert Kilduff, John Kienle, Connor McStay, Andrew Figueroa, Matthew Connolly, Edie Fost, Gina Roma, Jake Fonseca, Ido Levy, Michele Payne, Ryan Schenkel, Amir Malki, Lion Schneider, Aniruddha Narkhede, Shekeba Moshref, Alexandra Kisin, Olga Dodin, Bill Rippon, Henry Wrieth, John Ganci, Johnny Colino, Donna Habeger-Rose, Rakesh Pandey, Aditya Gidh, Aditya Gaur, Dennis Patterson, Samsuddin Salmani, Rambilas Varma, Rumana Rumana, Shubham Sharma, Aditya Gaur, Mayank Mishra, Rameswar Panda, Aditya Prasad, Matt Stallone, Gaoyuan Zhang, Yikang Shen, David Cox, Ruchir Puri, Dakshi Agrawal, Drew Thorstensen, Joel Belog, Brent Tang, Saurabh Kumar Gupta, Amitabha Biswas, Anup Maheshwari, Eran Gampel, Jason Van Patten, Matthew Runion, Sai Kaki, Yigal Bogin, Brian Reitz, Steve Pritko, Shahan Najam, Surya Nambala, Radhika Chirra, Rick Welp, Frank DiMitri, Felipe Telles, Amilcar Arvelo, King Chu, Ed Seminaro, Andrew Schram, Felix Eickhoff, William Hanson, Eric Mckeever, Michael Light, Dinakaran Joseph, Piyush Chaudhary, Piyush Shivam, Puneet Chaudhary, Wesley Jones, Robert Guthrie, Chris Bostic, Rezaul Islam, Steve Duersch, Wayne Sawdon, John Lewars, Matthew Klos, Michael Spriggs, Bill McMillan, George Gao, Ashish Kamra, Gaurav Singh, Marc Curry, Tushar Katarki, Joe Talerico, Zenghui Shi, Sai Sindhur Malleni, and Erwan Gallen. The infrastructure powering ibm's gen ai model development, 2025.

[7] Google. T5 11b. `https://huggingface.co/google-t5/t5-11b`, 2025.

[8] GPUDirect Storage: A Direct Path Between Storage and GPU Memory. `https://developer.nvidia.com/blog/gpudirect-storage/`.

[9] Mark Hildebrand, Jawad Khan, Sanjeev Trika, Jason Lowe-Power, and Venkatesh Akella. Autotm: Automatic tensor movement in heterogeneous memory systems using integer linear programming. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 875–890, 2020.

[10] 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. *arXiv preprint arXiv:2106.09685*, 2021.

 https://doi.org/10.52202/085713-1653

[11] Chien-Chin Huang, Gu Jin, and Jinyang Li. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1341–1355, 2020.

[12] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[13] IBM. Ibm granite. `https://huggingface.co/ibm-granite`, 2025.

[14] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. Basic performance measurements of the intel optane dc persistent memory module. *arXiv preprint arXiv:1903.05714*, 2019.

[15] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

[16] Hongsun Jang, Jaeyong Song, Jaewon Jung, Jaeyoung Park, Youngsok Kim, and Jinho Lee. Smart-infinity: Fast large language model training using near-storage processing on a real system. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 345–360. IEEE, 2024.

[17] Jaehoon Jung, Jinpyo Kim, and Jaejin Lee. Deepum: Tensor migration and prefetching in unified memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 207–221, 2023.

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[19] Youngeun Kwon and Minsoo Rhu. Beyond the memory wall: A case for memory-centric hpc system for deep learning. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*, Fukuoka, Japan, 2018.

[20] Tung D Le, Haruki Imai, Yasushi Negishi, and Kiyokuni Kawachiya. Tflms: Large model support in tensorflow by graph rewriting. *arXiv preprint arXiv:1807.02037*, 2018.

[21] Shaobo Li, Yirui Eric Zhou, Yuqi Xue, Yuan Xu, and Jian Huang. Managing scalable direct storage accesses for gpus with gofs. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, page 979–995, New York, NY, USA, 2025. Association for Computing Machinery.

[22] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, et al. Torchtitan: One-stop pytorch native solution for production ready llm pre-training. *arXiv preprint arXiv:2410.06511*, 2024.

[23] Changyue Liao, Mo Sun, Zihan Yang, Kaiqi Chen, Binhang Yuan, Fei Wu, and Zeke Wang. Adding nvme ssds to enable and accelerate 100b model fine-tuning on a single gpu. *arXiv preprint arXiv:2403.06504*, 2024.

[24] Jonas Markussen, Lars Bjørlykke Kristiansen, Pål Halvorsen, Halvor Kielland-Gyrud, Håkon Kvale Stensland, and Carsten Griwodz. Smartio: Zero-overhead device sharing through pcie networking. *ACM Transactions on Computer Systems*, 38(1–2), jul 2021.

[25] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[26] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[27] Apoorve Mohan, Robert Walkup, Bengi Karacali, Ming-hung Chen, Abdullah Kayi, Liran Schour, Shweta Salaria, Sophia Wen, I-hsin Chung, Abdul Alim, Constantinos Evangelinos, Lixiang Luo, Marc Dombrowa, Laurent Schares, Ali Sydney, Pavlos Maniotis, Sandhya Koteshwara, Brent Tang, Joel Belog, Rei Odaira, Vasily Tarasov, Eran Gampel, Drew Thorstensen, Talia Gershon, and Seetharami Seelam. *Vela: A Virtualized LLM Training System with GPU Direct RoCE*, page 1348–1364. Association for Computing Machinery, New York, NY, USA, 2025.

[28] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[29] Xiaonan Nie, Yi Liu, Fangcheng Fu, Jinbao Xue, Dian Jiao, Xupeng Miao, Yangyu Tao, and Bin Cui. Angel-ptm: A scalable and economical large-scale pre-training system in tencent. *arXiv preprint arXiv:2303.02868*, 2023.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[31] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[32] Bharadwaj Pudipeddi, Maral Mesmakhosroshahi, Jinwen Xi, and Sujeeth Bharadwaj. Training large neural networks with constant memory using a new execution algorithm. *arXiv preprint arXiv:2002.05645*, 2020.

[33] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, C. J. Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-mei Hwu. Gpu-initiated on-demand high-throughput storage access in the bam system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 325–339, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, Chris J Newburn, Dmitri Vainbrand, I-Hsin Chung, et al. Gpu-initiated on-demand high-throughput storage access in the bam system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 325–339, 2023.

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[37] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, St. Louis, Missouri, 2021.

[38] Jie Ren, Jiaolin Luo, Kai Wu, Minjia Zhang, Hyeran Jeon, and Dong Li. Sentinel: Efficient tensor migration and allocation on heterogeneous memory systems for deep learning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 598–611. IEEE, 2021.

[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, 2021.

 https://doi.org/10.52202/085713-1653

[40] 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. *CoRR*, abs/2101.06840, 2021.

[41] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[42] Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, et al. Building ai agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pages 99–110, 2024.

[43] Xiaoyang Sun, Wei Wang, Shenghao Qiu, Renyu Yang, Songfang Huang, Jie Xu, and Zheng Wang. Stronghold: fast and affordable billion-scale deep learning model training. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17. IEEE, 2022.

[44] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pages 41–53, 2018.

[45] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. {dLoRA}: Dynamically orchestrating requests and adapters for {LoRA}{LLM} serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 911–927, 2024.

[46] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. Netllm: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 661–678, 2024.

[47] Kun Wu, Jeongmin Brian Park, Xiaofan Zhang, Mert Hidayetoğlu, Vikram Sharma Mailthody, Sitao Huang, Steven Sam Lumetta, and Wen-mei Hwu. Tba: Faster large language model training using ssd-based activation offloading. *arXiv preprint arXiv:2408.10013*, 2024.

[48] Tailing Yuan, Yuliang Liu, Xucheng Ye, Shenglong Zhang, Jianchao Tan, Bin Chen, Chengru Song, and Di Zhang. Accelerating the training of large language models using efficient activation rematerialization and optimal hybrid parallelism. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 545–561, 2024.

[49] Haoyang Zhang, Yirui Zhou, Yuqi Xue, Yiqi Liu, and Jian Huang. G10: Enabling an efficient unified gpu memory and storage architecture with smart tensor migrations. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–410, 2023.

[50] Jie Zhang and Myoungsoo Jung. Zng: Architecting gpu multi-processors with new flash for scalable data analysis. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1064–1075. IEEE, 2020.

[51] Jie Zhang, Miryeong Kwon, Hyojong Kim, Hyesoon Kim, and Myoungsoo Jung. Flashgpu: Placing new flash next to gpu cores. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[52] Haoran Zhou, Wei Rang, Hongyang Chen, Xiaobo Zhou, and Dazhao Cheng. Deeptm: Efficient tensor management in heterogeneous memory for dnn training. *IEEE Transactions on Parallel and Distributed Systems*, 2024.

We organize our appendix as follows:

1. Appendix A provides an extended discussion of related work.
2. Appendix B provides an detailed settings of the ZeRO baseline used in our evaluation.
3. Appendix C provides a set of additional results.
4. Appendix D discusses the limitation of our work.
5. Appendix E lists the licenses of code and datasets we used in our work.

A Related Work

A.1 Expanding GPU Memory

To overcome the GPU memory wall issue, researchers have expanded GPU memory with CPU memory, persistent memory, and SSDs. Previously, when models were smaller, expanding GPU memory solely with CPU memory [17, 39, 11, 44, 32, 41, 43, 20] is reasonable. However, expanding GPU memory with host CPU memory only is expensive. As recent LLMs become increasingly large, CPU memory cannot provide enough capacity. Therefore, researchers have proposed to use persistent memory [14] to expand GPU memory [38, 9, 52]. Alternatively, to obtain even larger capacity, SSDs [1, 49, 37, 16, 23, 47, 51, 29, 50] are also used to expand GPU memory.

Compared to existing studies, TERAIO utilizes both SSDs and CPU memory for tensor migration. It allows users to flexibly define the maximum CPU memory used for LLM training, our lifetime-aware tensor migration algorithm helps achieve better training performance by generating optimized tensor migration plans.

A.2 Memory Efficient LLM Training

Besides tensor offloading discussed in this paper, many memory-efficient training techniques are widely used in pre-training and fine-tuning, such as activation checkpointing [15, 12, 48, 2], Low-Rank Adaptation (LoRA) [10, 45]. Activation checkpointing balances memory consumption and computation time by partially recomputing the forward propagation during backward propagation, which slows down training. In contrast, TERAIO enables LLM training without recomputation. LoRA reduces the number of trainable parameters during LLM fine-tuning. TERAIO can be combined with LoRA to further reduce memory consumption and is also suitable for pre-training.

A.3 LLM Training Infranstructure

Recent studies have developed large-scale training infrastructures for LLMs. For example, Vela [27] introduces a virtualized training system that employs GPUDirect over RoCE networks to reduce communication overhead and improve scalability across multi-node clusters. IBM's Gen AI training platform [6] provides a production-ready stack with distributed scheduling, optimized I/O pipelines, and efficient resource management for foundation model training.

These systems focus primarily on distributed orchestration and communication efficiency. TERAIO is orthogonal to these works and can be seamlessly integrated with such infrastructures to alleviate GPU memory limitations. By enabling smart tensor migration, TERAIO enhances memory scalability without requiring changes to communication layers or cluster management frameworks.

B ZeRO Settings

We show the performance-critical parameters in the table below. With these settings, we ensure ZeRO-Infinity achieves reasonable performance with our hardware setup.

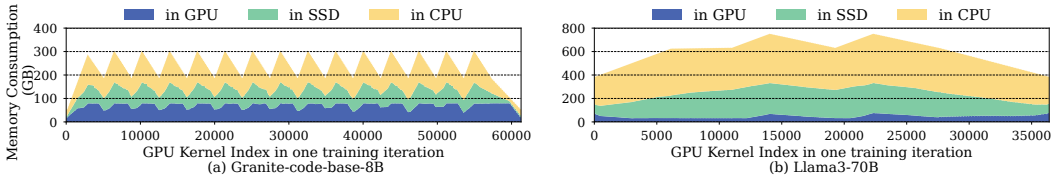
We enabled the `pipeline_read/write` parameters to optimize computation and data I/O overlap during optimizer state updates. We tuned parameters `pin_memory`, `buffer_count`, and `buffer_size` to optimize tensor offloading throughput. For `param_persistence_threshold` and `model_persistence_threshold`, we use their default values.

 https://doi.org/10.52202/085713-1653

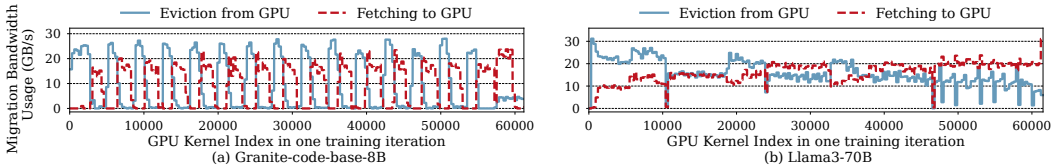
Table 4: Detailed ZeRO settings used in evaluation.

Parameter	Value	Description
`stage`	3	Uses Zero-3
`pipeline_read/write`	`True`	Overlaps read/write of next/previous tile with computation of current tile
`pin_memory`	`True`	Uses page-locked CPU memory for faster transfers
`buffer_count`	4	Number of async I/O buffers for optimizer states
`buffer_count`	18	Number of async I/O buffers for parameters
`buffer_size`	300/540M	Size of each parameter buffer
`param_persistence_threshold`	100K	Do not partition parameters smaller than this
`model_persistence_threshold`	`sys.maxsize`	Upper bound of unpartitioned parameters

C Additional Evaluation Results



(1) Memory usage breakdown of the program during training.



(2) Average migration bandwidth usage during training.

Figure 10: The memory usage breakdown and average migration bandwidth utilization when training the Granite-code-base-8B model and the Llama3-70B model in TERAIO-Mixed.

Migration bandwidth utilization. Figure 10 shows that TERAIO maintains high utilization of bidirectional migration bandwidth for most of the training time, thanks to our I/O-aware migration algorithm. This is particularly true when we train large models that require more memory and higher tensor migration traffic. Specifically, Granite and Llama3-70B utilize more than 16 GB/s migration bandwidth for more than 64% and 82.3% of the training time, respectively.

Host resource utilization. GDS provides significant scalability advantages by eliminating host resource contention. Without GDS, tensor offloading frameworks suffer from scalability bottlenecks. We validate this by measuring host resource usage as we scale the number of GPUs and SSDs.

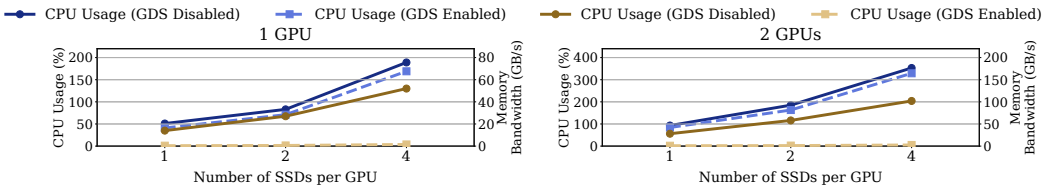


Figure 11: Host resource utilization. The measured CPU usage is relative to a single core.

As shown in Figure 11, host CPU usage increases linearly with the increasing number of GPUs and SSDs. With 2 GPUs and 4 SSDs per GPU, over 100 GB/s of host memory bandwidth is consumed, and more than 3 cores are fully used. This is because host memory is used as a bounce buffer for data transfers between GPUs and SSDs. When GDS is enabled, the host CPU consumption is significantly reduced. The host CPU utilization is reduced by 12.3% and host memory bandwidth

usage is decreased by 97.4% on average. As TERAIO aims to achieve low-cost LLM training, it is critical to minimize the use of host CPU and memory resources.

D Limitations

Our current implementation of GDS uses NVIDIA's cuFile library which still relies on the host file system to manage SSDs (e.g., filesystem metadata operations). Recent GDS studies such as GPU-initialized storage [24, 34] and GoFS [21] allow the GPU to fully bypass the host CPU by moving both the control path and data path for interacting with SSDs to the GPU. We wish to employ these studies to further reduce the overheads at the host side.

Although NVLink-C2C in NVIDIA Grace systems provides high GPU-CPU bandwidth, the memory capacity is still hard to scale, due to the fundamental DRAM scaling walls (physical limitations, power wall, and architecture constraints). As the memory capacity is insufficient for large models, expanding the GPU/host memory with low-cost and scalable SSDs ($0.2/GB for SSDs vs. $4/GB for DRAM on average) has become a practical and promising solution. Even if the cost is not a concern, our lifetime-aware tensor offloading approach can be applied to new and emerging memory technologies such as CXL memory. By expanding the GPU/host memory with external memory devices via CXL, new performance tradeoffs need to be considered, as the bandwidth and latency of accessing CXL memory vs. SSDs are different. We wish to extend TERAIO with new memory technologies as future work.

E Code and Datasets License

Codebase. Our model implementation is based on the Huggingface transformer repo (Apache-2.0 License) and the Torchtitan repo (BSD 3-Clause License).

Our framework implementation is based on Pytorch (BSD 3-Clause License).

The ZeRO series baselines we evaluated are from Megatron-DeepSpeed repo (Apache 2.0 License).

Datasets. We train the models using the C4 dataset (ODC-BY License).

 https://doi.org/10.52202/085713-1653

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction accurately summarize the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of our work in Appendix D.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.

- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: No theory is included in the paper.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We introduced detailed models, hardware configurations, ML frameworks, and baselines in §4.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.

https://doi.org/10.52202/085713-1653

- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We submit the code in the supplemental materials.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We presented all the training hyperparameters in §4.1. The hardware configuration for training all baselines are listed in Table 2. For each batch size and sequence length, we select the best hyperparameter for each baseline. We use AdamW as the optimizer and C4 as the dataset for all training tasks.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Training LLMs incurs significant costs. Although we did not train for hundreds of iterations, we conducted experiments across multiple LLMs, batch sizes, sequence lengths, and memory and storage configurations. The consistent results across these settings serve as a form of repeated evaluation and support the robustness of our findings.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper provides information on the compute resources in §4.1.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

https://doi.org/10.52202/085713-1653

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We checked that our paper conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: For organizations and companies with ample GPU resources, our work enables training or fine-tuning LLMs with larger batch sizes, leading to better GPU utilization. For smaller organizations and educational institutions that previously faced barriers due to limited resources, our work lowers the cost of access to LLMs.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No model or dataset is released in this paper.

    Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We included the license for the used code and datasets in Appendix E.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

https://doi.org/10.52202/085713-1653

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.