# An Efficient Heterogeneous Co-Design for Fine-Tuning on a Single GPU

Ruijia Yang
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
ryang379@connect.hkust-gz.edu.cn

Zeyi Wen*
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
wenzeyi@hkust-gz.edu.cn

## Abstract

Fine-tuning Large Language Models (LLMs) has become essential for domain adaptation, but its memory-intensive property exceeds the capabilities of most GPUs. To address this challenge and democratize LLM fine-tuning, we present SlideFormer, a novel system designed for single-GPU environments. Our innovations are: (1) A lightweight asynchronous engine that treats the GPU as a sliding window and overlaps GPU computation with CPU updates and multi-tier I/O. (2) A highly efficient heterogeneous memory management scheme significantly reduces peak memory usage. (3) Optimized Triton kernels to solve key bottlenecks and integrated advanced I/O. This collaborative design enables fine-tuning of the latest 123B+ models on a single RTX 4090, supporting up to 8× larger batch sizes and 6× larger models. In evaluations, SlideFormer achieves 1.40× to 6.27× higher throughput while roughly halving CPU/GPU memory usage compared to baselines, sustaining >95% peak performance on both NVIDIA and AMD GPUs.

## CCS Concepts

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Computing methodologies** → *Machine learning*.

## Keywords

MLSys, LLM fine-tuning, Heterogeneous memory management

**ACM Reference Format:**
Ruijia Yang and Zeyi Wen. 2026. An Efficient Heterogeneous Co-Design for Fine-Tuning on a Single GPU. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing with their remarkable capabilities across diverse tasks [20, 26], and fine-tuning open-source pre-trained models [1, 2, 25] on specific datasets is often preferred over training from scratch to achieve specialized performance [34]. However, as the models continue to grow in size, their fine-tuning memory requirements increase linearly. For example, fine-tuning an 8B model with mixed precision training [21] requires over 128 GB of GPU memory, far exceeding the VRAM of most high-end GPUs (e.g., 24-96 GB).

This memory bottleneck prevents the democratization of LLM fine-tuning, posing a significant barrier for individuals and small labs without access to GPU clusters or cloud resources. For single-GPU scenarios, a paradox arises: modern GPUs such as RTX 4090 possess ample computational power to fine-tune an 8B model, yet existing methods cannot efficiently handle the bottleneck, creating an urgent need for single-GPU solutions that break the VRAM wall.

A key trend motivating our work is the increasingly divergent growth trajectories between CPU and GPU memory, as shown in figure 1. Consumer systems now utilize DDR5 memory with doubled capacity (up to 256 GB) and faster I/O (PCIe and NVMe), whereas the maximum VRAM on GPUs has seen modest increases, from 24 GB (RTX 3090) in 2020 to 32 GB (RTX 5090) by 2025. This widening gap makes offloading attractive, turning single-GPU fine-tuning into a heterogeneous system design problem: How can we holistically co-design a system to leverage the entire platform (GPU, CPU, RAM, NVMe) to overcome the VRAM bottleneck?

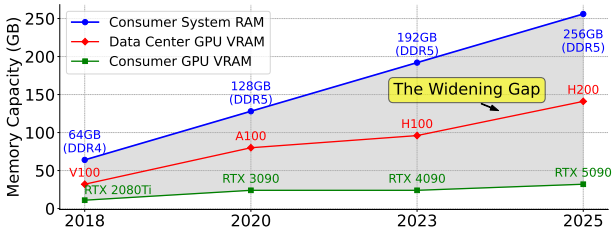


**Figure 1: The widening gap between CPU and GPU memory.**

Various methods have been proposed to address the memory constraints in LLM fine-tuning. Distributed techniques such as Pipeline Parallelism [11, 22], Tensor Parallelism [30], and Data Parallelism [16, 27] are generally unsuitable for single-GPU scenarios. Parameter-efficient fine-tuning [19] methods such as LoRA [10] have been proven insufficient to match the performance of full parameter fine-tuning in many cases [31]. Among existing offloading systems, ZeRO-Offload [29] and ZeRO-Infinity [28] are widely recognized. However, their designs are primarily for multi-GPU settings and fail to effectively pipeline computation with transfers and CPU updates, leaving significant room for performance improvement in single-GPU scenarios. Although some works [12, 17, 32] have explored this overlap potential, they are incompatible with recent LLMs and lack fine-grained optimizations for memory and efficiency, which are critical for practical usability.

*Corresponding author: wenzeyi@hkust-gz.edu.cn


*DAC '26, Long Beach, CA, USA*

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

To address the challenge, we present SlideFormer, a novel framework optimized for single-GPU fine-tuning through holistic heterogeneous co-design. Our work makes the following contributions:

- **A Lightweight Asynchronous Engine:** We propose a Layer-Sliding architecture that maintains a small, active window on the GPU, orchestrated by a multi-pipeline engine built on a lightweight thread-based mechanism, which efficiently overlaps GPU computation with CPU updates and I/O across hierarchies.
- **Efficient Heterogeneous Memory Management:** A queue of pre-allocated GPU cache units eliminates fragmentation and reallocation, while host-side shared buffers for gradients and type conversion reduce peak CPU memory by over 25%. In concert with our pipeline, this co-design enables fine-tuning with significantly less GPU and CPU memory than prior work.
- **Integrated Advanced I/O and Optimized Kernels:** We extend the memory hierarchy to NVMe and pioneer the integration of GPUDirect Storage[4] for offloading, bypassing the CPU. We also integrate a suite of fused Triton kernels for computations, resolving critical memory bottlenecks overlooked by previous systems.

The holistic co-design translates directly to state-of-the-art performance and scalability, enabling fine-tuning >123B models on a single RTX 4090. For a high-end PC equipped with 256 GB CPU memory, models up to 24B can be fine-tuned at over 95% peak GPU performance on both NVIDIA and AMD GPUs. Compared to existing frameworks, SlideFormer achieves a 1.40× to 6.27× improvement in throughput, reduces GPU memory consumption by over 50%, lowers CPU memory usage by approximately 40%, and supports 8x larger batch sizes and 6× larger model sizes.

Our work is implemented based on PyTorch [16] and Transformers [35] libraries, ensuring compatibility with the latest model architectures (e.g., Llama, Qwen). We expect SlideFormer to democratize LLM fine-tuning, enabling individuals and researchers with limited resources to leverage the power of large models.

## 2 Background

### 2.1 Memory Challenges in LLM Fine-Tuning

Fine-tuning adapts a pre-trained LLM to a target domain with far fewer steps and data than pre-training; yet it remains memory-bound at scale. For a model with $N$ parameters and $n$ layers, hidden size $h$, sequence length $s$, and batch size $b$, memory demand comes from: parameters, gradients, optimizer states, and activations.

**Static footprints.** Parameters are typically stored in `FP16/BF16` ($2N$ bytes), while gradients contribute another $2N$ in `FP16/BF16`. The Adam [13] optimizer is commonly used; it adds two `FP32` states per parameter (momentum/variance, $8N$), making optimizer states the largest static term. Besides, mixed-precision training [21] requires the optimizer to maintain an `FP32` master copy ($4N$) of parameters for stability. Forward activations scale with $O(n \cdot h \cdot s \cdot b)$ and must be available for the backward pass unless recomputed. A succinct approximation is:

$$Mem_{req} = \underbrace{2N}_{\text{Params}} + \underbrace{2N}_{\text{Grads}} + \underbrace{4N + 8N}_{\text{Optimizer States}} + \underbrace{O(n \cdot h \cdot s \cdot b)}_{\text{Activations}} \quad (1)$$

**Single-GPU tension.** Distributed parallelism techniques [11, 16, 22, 27, 30] amortize memory across multiple devices but are infeasible on a single GPU. A single high-end GPU has ample compute to fine-tune multi-billion-parameter models; yet the footprint in Eq. (1) frequently exceeds VRAM, forming the central bottleneck.

**Common mitigations.** *Gradient checkpointing* [3] trades 30% extra compute for >80% activation savings; *PEFT* (e.g., Adapter [8], LoRA [10]) updates a small subset of weights but underperforms compared to full-parameter fine-tuning on domain-critical tasks [18, 19, 34, 36]; *Kernel optimizations* (e.g., FlashAttention [6], xFormers [14], Liger [9]) reduce transient allocations and improve throughput. These techniques are complementary, but not sufficient to resolve the VRAM wall in single-GPU full-parameter fine-tuning.

### 2.2 Existing Offloading Techniques

A key trend driving us is the increasingly divergent growth trajectories between CPU and GPU memory, as shown in Figure 1. Recent PCs and workstations with abundant CPU memory (e.g., up to 256 GB DDR5) and high-speed NVMe storage enable memory-efficient LLM fine-tuning through strategic offloading. Coupled with faster PCIe interconnects, stronger CPU performance, and technologies like GPUDirect Storage [4], this motivates a pipeline-aware offloading design that jointly orchestrates the GPU, CPU, and NVMe rather than treating VRAM as the only limiting resource.

Several representative frameworks have been developed to this end. ZeRO-Offload [29] pioneers offloading the optimizer and gradients to the CPU. Then ZeRO-Infinity [28] extends it to a multi-tiered memory system, dynamically offloading components to both CPU and NVMe. Other notable systems, such as Transformer Engine [24] and NeMo [23], provide a layer-wise approach for activation offloading, and ColossalAI [15]'s *gemini* [7] introduces a dynamic chunk-based hetero-memory management. Besides, several research prototypes have explored similar concepts [12, 17, 32].

### 2.3 Limitations of Existing Solutions

While mainstream frameworks excel in distributed and multi-GPU settings, their design is not holistically co-designed for single-GPU scenarios. For instance, ZeRO-Offload and ZeRO-Infinity inherit considerable overhead from their distributed-first architecture; mechanisms intended for multi-GPU communication remain active on a single device, introducing additional memory footprint and latency. This, combined with underutilized CPU memory pools, creates significant overhead, as observed in Section 4.3. Similarly, ColossalAI's chunk-wise memory management, while effectively utilizing memory for larger models, is suboptimal for single-GPU efficiency. Critically, their design is synchronous at the update stage, leaving the GPU idle while waiting for the CPU update to finish.

Academic prototypes that recognize this overlap potential still suffer from critical design flaws. Stronghold [32] was an early attempt but relied on an outdated version of Megatron [30] and did not fully recognize or optimize for single-GPU environments. LoHan [17], a recent work, employs a multiprocess-based engine for asynchronous updates, which incurs IPC overhead, rather than a thread-based approach. Furthermore, LoHan utilizes on-demand memory management, which is prone to runtime fragmentation, and operates at a Param Group granularity without analyzing how to set its size. Its design choices are architecturally distinct from SlideFormer's pre-allocated and layer-granular design. These limitations, combined with incomplete optimizations (e.g., ignoring

the CrossEntropyLoss bottleneck) and limited model support (e.g., only GPT-2), necessitate a new, holistically designed system.

## 3 System Design

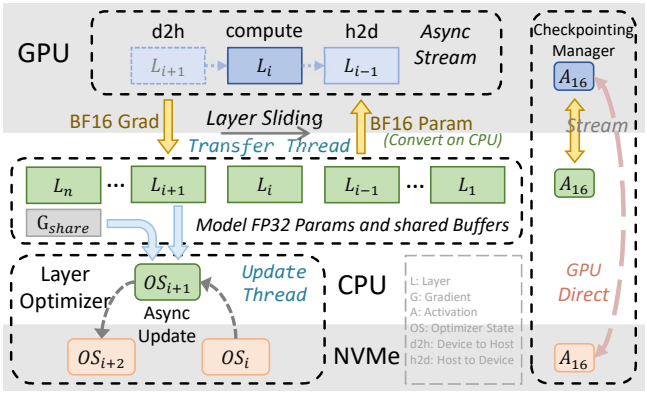


**Figure 2: Overview of SlideFormer.**

The design goal of SlideFormer is to break the memory wall of single-GPU fine-tuning through a holistic system-level co-design, while achieving state-of-the-art efficiency. We propose a unified architecture where computation scheduling, memory management, and I/O are jointly optimized. As illustrated in Figure 2, our system is built on three pillars: (1) a Layer-Sliding Architecture powered by a lightweight asynchronous engine, (2) a Pre-allocated Heterogeneous Memory system to eliminate overhead, (3) an Integrated I/O and Compute stack utilizing GPUDirect and fused kernels.

### 3.1 The Layer-Sliding Architecture

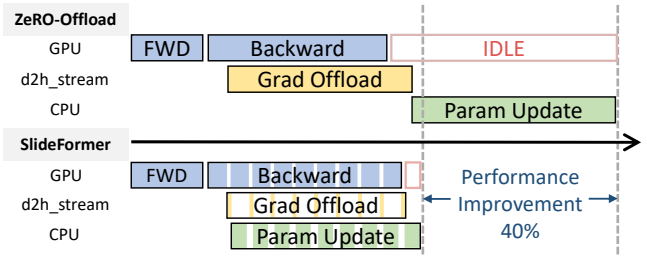


**Figure 3: Backward overlaps with parameter updates.**

**Asynchronous Parameter Updating:** As illustrated in Figure 3, we adopt a layer-granular approach to pipeline the backward and update with offloading. Once the backward computation for layer $L_i$ finishes on the GPU, its gradients $G_i$ are asynchronously transferred to host memory (d2h). In parallel, the CPU applies the optimizer to update $P_i$ using the host-resident optimizer states. While the CPU updates $P_i$, the GPU continues computing the backward pass for $L_{i-1}$ and prefetches the parameters for $L_{i-2}$ (h2d). This schedule eliminates the heterogeneous resource idle issue in ZeRO-Offload [29] by overlapping GPU-bound compute with CPU-bound updates and cross-tier transfers.

**Rationale for Layer Granularity:** The cornerstone of efficiency lies in our layer-granular strategy for memory management and computation scheduling, which restructures the fine-tuning process to maximize Hetero-hardware utilization. Layer is the smallest constitutional repeating unit in LLMs. Non-repeating units, such as the param-group used in ZeRO-Offload or LoHan [17], introduce complex management for various-sized components and require manual configuration. Critically, a multi-layer window is counterproductive in memory-constrained environments, as layers are computed serially, consuming scarce VRAM that could be used to increase the batch/model size while offering negligible benefits. As shown in Figure 4, the critical batch size required to achieve effective overlap remains remarkably stable across different layer sizes (from 77M-3B to 878M-72B). Because all backward pipeline latencies ($T_{bwd}$, $T_{grad\_d2h}$, $T_{update}$) scale proportionally with granularity, the overlap condition mainly depends on the batch size. A single layer is sufficient to saturate modern GPU, as evidenced by the high GPU utilization in Table 1 and Figure 8.

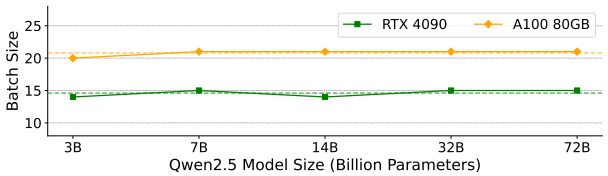


**Figure 4: Critical batch size for achieving full backward overlap with updates ($T_{bwd} \geq T_{grad\_d2h} + T_{update}$).**

**Thread-Based Lightweight Engine:** The backbone of SlideFormer's efficiency is its extensive use of asynchronous operations to overlap data transfers and CPU computations with the GPU workload. Unlike LoHan, which relies on a multi-process optimizer introducing IPC overhead, SlideFormer implements a lightweight thread-based engine through dedicated: (i) *CUDA Streams [5]*: Separate streams are employed for asynchronous h2d/d2h transfers and concurrent GPU computation. (ii) *CPU Threads*: Two thread executors, one for transfers between h2d/d2h and the other for Layer-Adam to update parameters, prevent potential blocking I/O or CPU-intensive tasks from stalling the main fine-tuning thread.

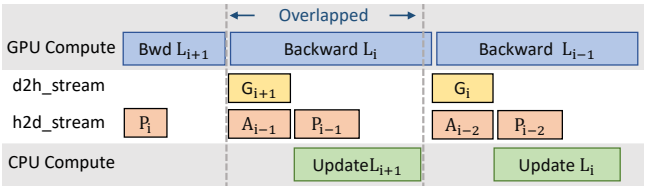


**Figure 5: Computation-communication overlap during backward propagation in GPU-CPU tier pipeline.**

**Condition for Effective Overlap:** The efficiency of our asynchronous engine hinges on latency hiding, where the following conditions should be met: (i) In forward pass, lossless overlap occurs when the computation time for the current layer is greater than or equal to the parameter prefetch time for the next layer, i.e., $T_{compute\_fwd} \geq T_{param\_h2d}$. (ii) In the backward pass, as illustrated in Figure 5, lossless overlap occurs when $T_{compute\_bwd} \geq T_{grad\_d2h} + T_{update}$. When NVMe offloading is enabled, the transfer overhead of the optimizer states makes $T_{update}$ the main performance bottleneck. To quantify the degree of backward overlap, we introduce the **hiding factor** ($\eta = T_{\text{bwd}}/(T_{\text{d2h}} + T_{\text{update}})$), where $\eta \geq 1$ indicates zero-overhead offloading. Table 1 presents the timeline breakdown for fine-tuning Qwen2.5-14B, confirming that our

architecture achieves effective overlap across various hardware. Unlike sequential methods such as ZeRO-Offload that would completely stall the GPU, SlideFormer maintains a robust performance advantage even on imbalanced hardware where full overlap ($\eta < 1$) is infeasible, using extremely powerful or memory-limited GPUs.

**Table 1: Profile timelines of backward stage for SlideFormer during the fine-tuning of Qwen2.5-14B. (All time in ms)**

| System | Batch Size | $T_{bwd}$ | $T_{d2h}$ | $T_{update}$ | Factor ($\eta$) | GPU Util. (%) |
|---|---|---|---|---|---|---|
| *RTX4090 24GB (PC)* | 16 | 170 | 22 | 175 | **0.66** | 93.1 |
| | 32 | 340 | 25 | 195 | **1.55** | 96.9 |
| | 64 | 660 | 25 | 195 | **3.00** | 98.4 |
| *A100 80GB (Server)* | 32 | 225 | 24 | 152 | **1.28** | 97.2 |
| | 64 | 450 | 25 | 151 | **2.56** | 98.8 |
| | 128 | 910 | 25 | 153 | **5.11** | 99.3 |

## 3.2 Efficient Heterogeneous Memory Co-Design

Previous works [17, 29, 32] often overlooked the evaluation and optimization of heterogeneous memory footprints, but we hope to co-design an extremely efficient, fixed footprint, and fragment free memory management system based on a layer sliding architecture.

**Pre-allocated GPU Cache Unit Queue:** Rather than keeping the entire model in GPU memory, SlideFormer maintains a window of active layers, which is exactly a queue of pre-allocated GPU cache units, each sized to hold a layer's parameters and gradients. During training, layers (i.e., parameters) sequentially slide into this cache queue to perform computations, after which the used units are released for new layers. Only during the backward pass, the gradients of each layer are offloaded to CPU memory. Unlike the *on-demand* allocation used by StrongHold [32] and LoHan [17], this unit reuse design ensures a fixed GPU memory footprint and avoids reallocation, reducing overhead and fragmentation.

**Optimized CPU Memory Layout with Shared Buffers:** On the CPU side, `FP32` parameter master copies of each layer are stored in a flattened, pinned tensor (`cpu_params_flat`) for efficient h2d transfers. To optimize memory usage, we employ shared buffers for intermediate data. Gradients offloaded from the GPU are stored in a layer-shared, pinned `BF16/FP16` tensor (`cpu_grad_flat`), which reduces the gradient footprint on CPU memory ($2N$ bytes) to $1/num\_layers$. Similarly, a layer-shared buffer is dedicated to convert `FP32` parameters to `BF16/FP16` before h2d transfer, thus avoiding additional transfer/memory costs of type conversion on the GPU and storing $2N$ bytes of `BF16/FP16` parameters in CPU memory. On the GPU side, parameters and gradients maintain `BF16/FP16` precision, following the mixed precision training [21] scheme.

**Sliding Activation**: To further alleviate GPU memory pressure from activations, we employ a sliding checkpointing mechanism modified from standard gradient checkpointing [3, 16]. After each layer's forward pass, activations are asynchronously offloaded to the CPU memory or NVMe and prefetched to the GPU memory for recomputation before the backward pass of that layer, ensuring that VRAM required for activations is limited to only a small window. We pre-allocate pinned tensors in CPU memory or files on SSDs for storing activations before the fine-tuning begins.

**Layer-Adam Optimizer**: A self-developed variant of DeepSpeed's CPU-Adam, it stores the optimizer states of each layer in a flattened tensor in the host memory. When the gradients of the layer are offloaded to the CPU, the optimizer updates the layer's parameters separately. Additionally, the optimizer states can be further offloaded to the NVMe tier, and an asynchronous offload-prefetch mechanism is established to reduce latency.

## 3.3 Integrated I/O and Compute Co-Design

The final pillar of our co-design optimizes the data movement paths and intra-layer computation to eliminate remaining bottlenecks that pure scheduling cannot address.

**GPUDirect Storage and NVMe Tiering:** To support models exceeding CPU RAM capacity, SlideFormer extends the memory hierarchy to NVMe storage. Crucially, we pioneer to integrate GPUDirect Storage (GDS) [4] for LLM fine-tuning offload. GDS establishes a direct data path between NVMe and GPU, bypassing the CPU bounce buffer. This "zero-copy" mechanism significantly reduces CPU utilization and PCIe bus contention, leaving CPU resources for asynchronous engine and parameter updates. We support offloading *activations* and *optimizer states* to this NVMe tier.

**Why Not Offload Parameters.** Although offloading parameters to NVMe storage could achieve lower memory usage and larger models, we deliberately avoid it due to diminishing returns: (i) Performance Degradation: Parameter transfers (h2d/d2h) are critical for overlapping with GPU computation (c.f. Section 3.1). Moving parameters to NVMe would shift the transfer bottleneck from PCIe to NVMe speed, severely hindering overall throughput. (ii) Simplified Data Paths: As shown in Figure 2, SlideFormer ensures that any given data type moves only between two memory tiers. Introducing NVMe as a third tier for parameters would complicate the data transfer path and add unnecessary overhead.

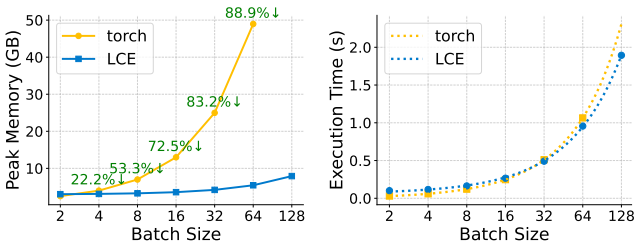


**Figure 6: Memory usage and execution time comparison between torch standard method and LCE for Llama3.1-8B.**

**Optimized Triton Kernels:** While our pipelines optimize *inter-layer* data movement, we integrate optimized Triton [33] kernels to accelerate *intra-layer* computational efficiency. Beyond FlashAttention [6], we employ efficient Triton kernels for operations like RoPE, RMSNorm, and SwiGLU, collectively reducing peak memory usage and improving throughput. Among these, the most critical optimization is the **fused LinearCrossEntropy kernel** for the output layer and loss computation, which addresses a major and often overlooked memory bottleneck. For recent models with large vocabularies like Llama-3.1, the intermediate logits tensor ($B \times S \times V$) can consume more VRAM than all preceding activations combined. LoHan [17] sidesteps this issue in evaluation by

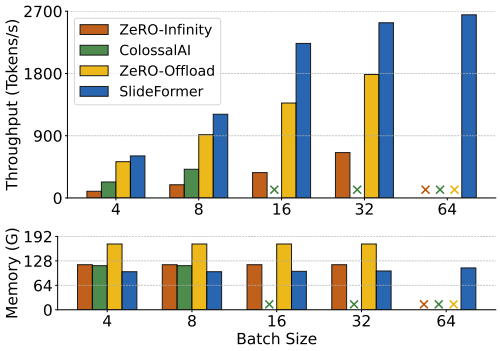


**Figure 7: Throughput and CPU memory comparison between SlideFormer and baselines for Llama-3.1-8B fine-tuning on RTX4090.**

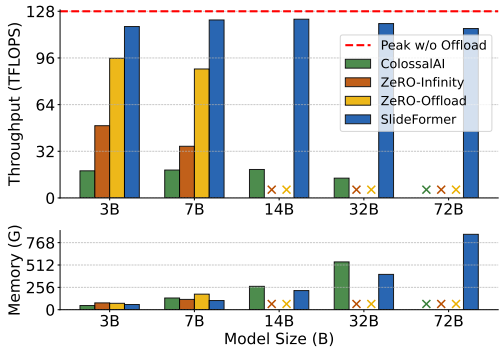


**Figure 8: Throughput and CPU memory comparison between SlideFormer and baselines for various sizes of Qwen2.5 on RTX4090.**

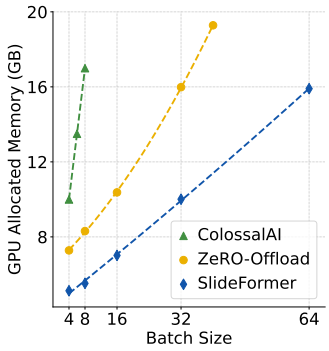


**Figure 9: GPU memory vs. batch size on various frameworks for Llama-3.1-8B.**

replacing the standard loss with MSE, which is impractical for real-world tasks. SlideFormer solves this directly by integrating a Fused LinearCrossEntropy (LCE) kernel. This kernel fuses the projection and loss calculation, computing gradients in small chunks to avoid materializing the full logits tensor. As shown in Figure 6, this reduces the memory footprint of the output layer by over 80% without sacrificing accuracy or speed, unlocking the ability to train with models and batch sizes essential for pipeline saturation.

## 4 Evaluation

In this section, we conduct a comprehensive evaluation of our design to demonstrate its performance and efficiency.

### 4.1 Experimental Setup

We evaluate SlideFormer on two types of platforms: a high-end PC (NVIDIA RTX 4090 24GB or AMD RX 7900XT 20GB, AMD Ryzen 9 9950X, 256GB DDR5) and a server (NVIDIA A100 80GB, dual Intel Xeon Gold 6338N, 1024GB DDR4). All experiments use PyTorch 2.7.0 and CUDA 12.5 with a fixed sequence length of 1024. For performance benchmarking, we use a synthetic dataset to ensure a consistent computational load (with a stable effective length).

We compare SlideFormer against leading offloading baselines: ZeRO-Offload [29], ZeRO-Infinity [28], ColossalAI [15], and LoHan [17]. To ensure a fair comparison, all frameworks use the latest versions with identical training configs, including activation checkpointing and optimized kernels where applicable. We evaluate a range of modern LLMs, including Llama-3.1 (8B) [1], Qwen-2.5 (3B-72B) [25], and Mistral (24B-123B) [2]. Performance is measured by Throughput (tokens/s and TFLOPS), peak Memory Usage (GPU and CPU), and trainable model size (B).

### 4.2 Throughput Scalability

SlideFormer demonstrates superior throughput scalability across both increasing batch sizes and model sizes, consistently outperforming leading offloading systems.

**Scalability with Batch Size.** As shown in Figure 7, SlideFormer outperforms all baselines across every batch size, achieving throughput improvements of 1.39×, 2.82×, 6.34× over baselines on Llama-3.1-8B. The results also illustrate our pipeline's dynamics: at smaller batch sizes, the step time remains constant, as the backward computation is insufficient to fully mask the update latency. However, as the batch size increases to 32, the system shifts to a compute-bound regime where the transfer and update latencies are effectively hidden. This, along with Figure 10, confirms our design's ability to leverage larger batch sizes for higher computational throughput.

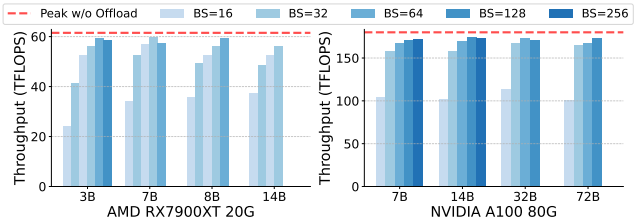


**Figure 10: The fine-tuning throughput of Qwen2.5 in various sizes on AMD RX7900XT and NVIDIA A100.**

**Scalability with Model Size.** Figure 8 show that SlideFormer not only delivers higher throughput than baselines at equivalent sizes but also dramatically extends the boundaries of trainable models on a single GPU. While ZeRO-Offload and ZeRO-Infinity fail to run models of 14B parameters or larger, SlideFormer successfully fine-tunes models exceeding 72B parameters. Crucially, SlideFormer's performance consistently reaches 90% to 95% of the peak non-offloading fine-tuning TFLOPS. This high utilization is robust across platforms, with Figure 10 confirming similar high efficiency (over 95% peak performance) on both AMD RX7900XT and NVIDIA A100 GPUs, underscoring SlideFormer's broad applicability.

### 4.3 Heterogeneous Memory Usage

SlideFormer's efficient control over memory across the hierarchy is what enables maximum scalability and batch sizes.

**CPU Memory Efficiency.** The lower panels of Figure 7 and Figure 8 illustrate that SlideFormer maintains the lowest CPU memory footprint across all scenarios, reducing usage by approximately 40% compared to the fastest baseline. This significant saving is a direct result of our optimized host memory layout, which utilizes layer-shared buffers for gradients and type conversion, eliminating redundant memory copies and peak consumption.

**GPU Memory Efficiency.** Figure 9 plots the GPU memory footprint against batch size, showing that SlideFormer consistently

uses the least VRAM, achieving a reduction of over 50% compared to ZeRO-Offload. This is attributed to our pre-allocated cache queue and the integrated Fused LCE kernel, which together alleviate the primary memory bottleneck in fine-tuning, making it feasible to train large models on consumer-grade hardware.

For example, an individual with a PC with 128GB CPU memory can fine-tune the Llama-3.1-8B model on a single RTX 4080 GPU. This is achievable on a single GPU without resorting to NVMe offloading, while maintaining nearly lossless throughput compared to non-offloaded training. This capability is a cornerstone of our goal to democratize access to large model fine-tuning.

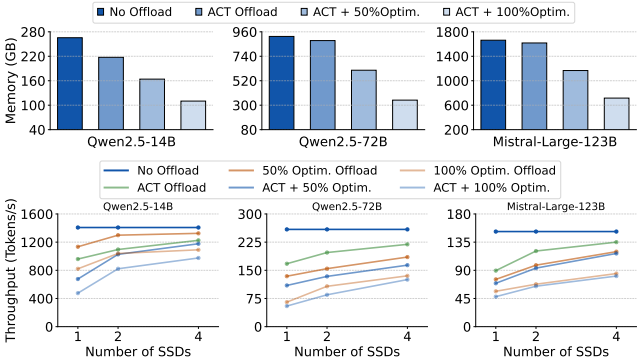


**Figure 11: Performance comparison of NVMe SSD counts.**

## 4.4 Analysis of NVMe Offloading

For models exceeding CPU memory capacity, SlideFormer leverages the optional NVMe tier. Activations and optimizer states can be offloaded asynchronously, with support for GPUDirect Storage and configurable offload fractions (50% or 100%) for optimizer states. Figure 11 illustrates the trade-off between the CPU memory savings achieved through various offloading strategies and the corresponding impact on throughput: First, performance scales near-linearly with the number of NVMe drives, as I/O bandwidth becomes the primary bottleneck. Second, by enabling all offloading options, SlideFormer can reduce CPU memory consumption by 60-80%, with a corresponding throughput degradation contained within 30-50%. Third, the optimal offloading strategy is model-size dependent. For smaller models like Qwen2.5-14B, activations constitute a larger portion of the offloaded data. Offloading them provides significant memory savings but incurs a notable performance penalty as it impacts both the forward and backward passes. In this case, offloading optimizer states alone yields a better performance-to-memory trade-off. Conversely, for larger models where optimizer states dominate the memory footprint, offloading them first is most effective, and the additional, marginal impact of offloading activations becomes negligible. We therefore recommend offloading activations only for the largest models or under severe CPU memory constraints.

## 4.5 Maximum Trainable Model Size

Figure 12 presents a comparison of the maximum model sizes that can be fine-tuned using SlideFormer versus baseline frameworks, and each point is derived from actual tests conducted on listed pre-trained models. The experimental results demonstrate that, unlike other baselines which are constrained by GPU memory and thus

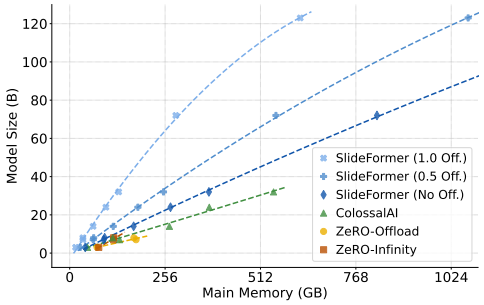


**Figure 12: Maximum trainable model size.**

limited in max trainable model size (e.g., Zero-offload supports up to 8B parameters, and ColossalAI supports up to 32B parameters), SlideFormer significantly extends the upper limit of fine-tunable model sizes. By shifting the primary memory constraint to CPU memory, SlideFormer enables the fine-tuning of models exceeding 123B parameters on a single GPU. For a high-end PC equipped with 256GB of CPU memory, enabling NVMe offloading allows fine-tuning models up to 90B parameters and can fine-tune models within 24B without throughput loss, as shown in Figure 8.

## 4.6 Compared to Related Works

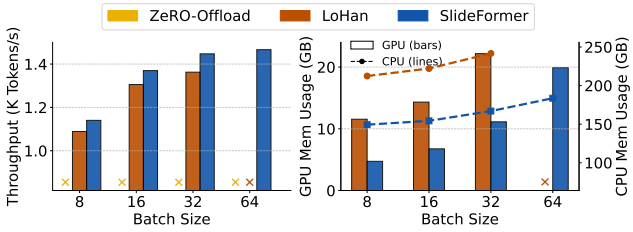


**Figure 13: Throughput and memory comparison between SlideFormer and LoHan for GPT2-13B on RTX4090.**

In recent research, LoHan [17] is one of comparable to our work. However, it only supports GPT-2 and uses a non-standard loss function (MSE) during evaluation to sidestep the associated GPU memory overhead. Figure 13 shows that under a standard GPT-2 Fine-tuning task, SlideFormer achieves superior performance, delivering higher throughput and consuming < 50% of the GPU memory and saving 30% in CPU memory usage. ZeRO-Offload failed to run due to exceeding GPU memory. This result fundamentally validates our better architecture design and memory management compared to LoHan, which make SlideFormer the current optimal co-designed solution for current single GPU fine-tuning tasks.

## 5 Conclusion

In this paper, we present SlideFormer, a novel system that implements a holistic heterogeneous co-design, which significantly enhances the efficiency of full-parameter LLM fine-tuning on a single GPU. SlideFormer achieves 1.40-6.27× throughput gains while substantially halving CPU/GPU memory usage. It enables training 6× larger models and handling 8× larger batch sizes, demonstrating high compatibility (over 95% peak performance on both NVIDIA&AMD GPUs) with the latest LLMs. The primary significance of SlideFormer is its democratization of LLM fine-tuning, empowering individual researchers and smaller organizations.

## References


[1] et al. Aaron Grattafiori. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[2] Mistral AI. 2024. Mistral-Large-Instruct-2411. https://huggingface.co/mistralai/Mistral-Large-Instruct-2411.

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[4] NVIDIA Corporation. 2021. NVIDIA GPUDirect Storage: Benchmarking and Configuration Guide. https://docs.nvidia.com/gpudirect-storage/.

[5] NVIDIA Corporation. 2025. CUDA Runtime API: Stream Management. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html.

[6] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[7] Jiarui Fang and Yang You. 2022. Meet Gemini: The Heterogeneous Memory Manager of Colossal-AI. https://colossalai.org/docs/advanced_tutorials/meet_gemini/.

[8] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*. PMLR, 2790–2799.

[9] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2024. Liger Kernel: Efficient Triton Kernels for LLM Training. *arXiv preprint arXiv:2410.10989* (2024). arXiv:2410.10989 [cs.LG] https://arxiv.org/abs/2410.10989

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

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

[12] Hongsun Jang, Jaeyong Song, Jaewon Jung, Jaeyoung Park, Youngsok Kim, and Jinho Lee. 2024. 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)*. IEEE, 345–360.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. 2022. xFormers: A modular and hackable Transformer modelling library. https://github.com/facebookresearch/xformers.

[15] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. 2023. Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training. In *Proceedings of the 52nd International Conference on Parallel Processing* (Salt Lake City, UT, USA) *(ICPP '23)*. Association for Computing Machinery, New York, NY, USA, 766–775. doi:10.1145/3605573.3605613

[16] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[17] Changyue Liao, Mo Sun, Zihan Yang, Jun Xie, Kaiqi Chen, Binhang Yuan, Fei Wu, and Zeke Wang. 2024. LoHan: Low-Cost High-Performance Framework to Fine-Tune 100B Model on a Consumer GPU. arXiv:2403.06504 [cs.DC] https://arxiv.org/abs/2403.06504

[18] Qijun Luo, Hengxu Yu, and Xiao Li. 2024. BAdam: A Memory Efficient Full Parameter Optimization Method for Large Language Models. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 24926–24958. https://proceedings.neurips.cc/paper_files/paper/2024/file/2c570b0f9938c7a58a612e5b00af9cc0-Paper-Conference.pdf

[19] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods. https://github.com/huggingface/peft.

[20] Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* 1 (2020), 3.

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

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

[23] NVIDIA. [n. d.]. NVIDIA/NeMo: A Scalable Generative AI Framework Built for Researchers and Developers Working on Large Language Models, Multimodal, and Speech AI (Automatic Speech Recognition and Text-to-Speech). https://github.com/NVIDIA/NeMo. Accessed: May 15, 2025, n.d..

[24] NVIDIA. 2024. Transformer Engine: A Library for Accelerating Transformer Models on NVIDIA GPUs. https://github.com/NVIDIA/TransformerEngine. Version 2.1.0, accessed on 2025-04-23.

[25] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

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

[27] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[28] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. 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*. 1–14.

[29] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. Zero-offload: Democratizing Billion-Scale Model Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 551–564.

[30] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[31] Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. 2025. LoRA vs Full Fine-tuning: An Illusion of Equivalence. arXiv:2410.21228 [cs.LG] https://arxiv.org/abs/2410.21228

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

[33] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (Phoenix, AZ, USA) *(MAPL 2019)*. Association for Computing Machinery, New York, NY, USA, 10–19. doi:10.1145/3315508.3329973

[34] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2021).

[35] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[36] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507* (2024).