Deferred Continuous Batching in Resource-Efficient Large Language Model Serving

Deferred Continuous Batching in Resource-Efficient Large Language Model Serving

Yongjun He
ETH Zürich
Zürich, Switzerland
yongjun.he@inf.ethz.ch

Yao Lu
National University of Singapore
Singapore
luyao@comp.nus.edu.sg

Gustavo Alonso
ETH Zürich
Zürich, Switzerland
alonso@inf.ethz.ch

Abstract

Despite that prior work of batched inference and parameter-efficient fine-tuning techniques have reduced the resource requirements of large language models (LLMs), challenges remain in resource-constrained environments such as on-premise infrastructures to serve workload that is composed of both inference and fine-tuning jobs. Prior solutions must either pause existing jobs which causes service interruptions, or queue new jobs which results in a long delay.

We present FineInfer, an efficient serving system that enables concurrent LLM fine-tuning and inference. FineInfer leverages base model multiplexing and a new task scheduling mechanism, namely deferred continuous batching, to enable iteration-level context switch and accelerate fine-tuning while offering inference latency that compromises service level agreements. Our evaluation shows that FineInfer outperforms prior solutions by up to 3x in fine-tuning latency, and 36x when the models are larger than the GPU memory.

CCS Concepts: • **Computing methodologies** → **Concurrent algorithms**; *Natural language generation*; • **Computer systems organization** → *Neural networks.*

Keywords: Systems for large language models, LLM fine-tuning, LLM inference

ACM Reference Format:
Yongjun He, Yao Lu, and Gustavo Alonso. 2024. Deferred Continuous Batching in Resource-Efficient Large Language Model Serving. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3642970.3655835

EuroMLSys '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0541-0/24/04…$15.00
https://doi.org/10.1145/3642970.3655835

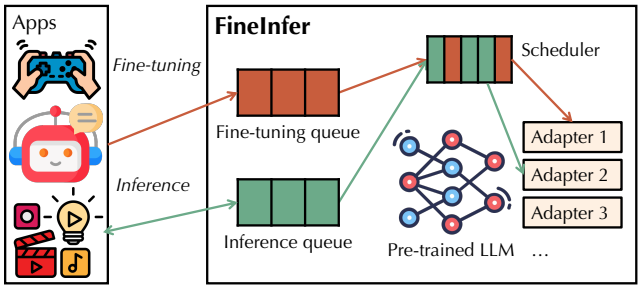

Figure 1. FineInfer offers the ability of LLM fine-tuning and inference at the same time.

1 Introduction

Large language models (LLMs) are becoming a critical building block in emerging applications such as chatbots, search engines, gaming, and translations [7]. Systems such as DeepSpeed [25] and Megatron [29] have enabled serving and fine-tuning LLMs, thus improving the accuracy of state-of-the-art pre-trained models in specific application scenarios.

When deploying LLMs in *resource-constrained environments* [24, 38] including laptops, dedicated GPU servers, and mobile devices[37], we observed new challenges to handle *heterogeneous* LLM workload that is composed of fine-tuning and inference jobs at the same time [3, 35]. For instance, after deploying a language model inference service for a chatbot, if the model requires an update to incorporate user feedback [14, 20], current systems must either terminate the inference service or route the fine-tuning job to other resources or time periods due to a lack of resources. On the contrary, when a fine-tuning job is in progress and a new model inference request comes in, current systems must dump the tuning checkpoint to memory or disk, and load the required model for inference. In both cases, a significant latency is incurred which has a negative impact on the service quality. The situation becomes worse if there are many models, applications and users.

We propose FineInfer, depicted in Figure 1, an LLM serving system that optimally orchestrates fine-tuning and inference jobs in resource-constrained environments. FineInfer leverages the following observations and techniques to improve upon state-of-the-art:

Firstly, with the advent of parameter-efficient fine-tuning (PEFT) techniques such as LoRA [10, 21], prefix tuning [19],

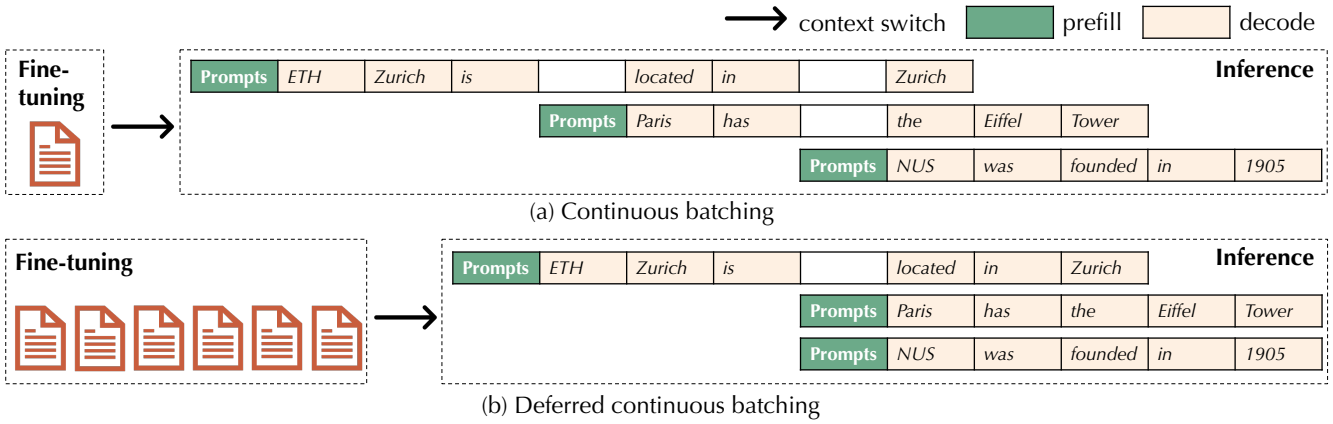

Figure 2. Illustration of different scheduling strategies.

and prompt tuning [15, 22], users may not need to tune the entire model which significantly saves time and resource. Take LoRA fine-tuning for example, given a pre-trained matrix $W \in \mathbb{R}^{d\times k}$, LoRA approximates the update during fine-tuning via low-rank decomposition $W' = W + \Delta W = W + BA$, where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$. Since the rank $r \ll min(d, k)$, the number of trainable parameters can often be 1000x smaller. Recently, Punica [8] and S-LoRA [27] offer highly efficient model serving by sharing the base LLM model in multiple LoRA fine-tuned applications; therefore, inference computation on the base model can be batched for improved throughput, while subsequent computation on multiple LoRA adapters can be accelerated via Multi-size Batched Gather Matrix-Vector Multiplication (MBGMV) [27] or Segmented Gather Matrix-Vector Multiplication (SGMV) [8]. FineInfer builds upon this idea to leverage base model multiplexing and fine-grained scheduling to enable fine-tuning and inference of many LoRA adaptors at the same time; doing so significantly minimizes the switching overhead in-between fine-tuning and inference.

Secondly, we propose *Deferred Continuous Batching*, a new strategy to schedule fine-tuning and inference at the granularity of each fine-tuning iteration, thus further batching up inference computation and reducing fine-tuning latency. Specifically, inference requests are pushed out slightly according to their latency service level agreement (SLA) to improve their chance of being able to batch up with other requests. We found this simple strategy to be surprisingly effective, and sometimes even critical. This is due to the fact that fine-tuning still demands a lot of resources and therefore will incur a significant latency penalty in the latter when sharing the GPU with inference jobs. However, with only 0.3-0.5 second deferred inference which is often not noticeable in end-to-end model serving applications, we observe a 26-36% improvement in the fine-tuning throughput.

Last but not least, in situations when LLMs do not fit in the GPU memory, deferred continuous batching kicks in again to reduce the frequency of parameter movement in-between CPU and GPU memory, thus improving the chance of compute batching and reducing overall latency. The key idea is similar to batch fine-tuning and inference inputs in the forward pass and update only adapters relevant to the fine-tuning tasks in the backward pass.

Evaluation on a variety of models and workloads using a single Nvidia 3090 GPU shows that FineInfer incurs very low switching overhead and that deferred continuous batching is highly beneficial for heterogeneous LLM workloads. For LLMs that fit into GPU memory, FineInfer can improve fine-tuning throughput by up to 3x compared to SoTA systems while meeting inference latency SLAs. For LLMs larger than GPU memory, we find that FineInfer can achieve up to 36x improvement compared to SoTA offloading-based systems. To verify the effectiveness of deferred continuous batching, we also implemented base model multiplexing and iteration-level context switching into other variants for fair comparison.

This paper introduces our preliminary ideas and early-stage efforts. FineInfer is open-sourced at https://github.com/llm-db/FineInfer. Contributions and feedback from the community are eagerly welcomed.

2 LLM Serving in Resource-Constrained Environments

LLM inference is an iterative autoregressive process where an output token is generated in each model iteration. The processing of an inference request consists of two stages: 1) The prefill stage first generates the KV cache tensors representing the context of the input prompts, and this phase is executed only once. 2) The decoding stage then utilizes the KV cache tensor to generate a new token, and the context of this token is also added to the KV cache tensor. This phase will be executed multiple times until the generation length limit is reached.

Table 1. The breakdown of switching overhead with Llama2-7B workloads. In each column, we report the overhead corresponding to fine-tuning (left) and inference (right) tasks.

Stage	DeepSpeed	Colossal-AI	FineInfer
Task initialization	1.153 / 0.015 s	0.28 / 0.045 s	0 / 0 s
Task cleanup	2.330 / 1.252 s	3.456 / 1.376 s	0 / 0 s
Data movement	5.882 s	5.918 s	0 - 0.052 s

Continuous batching. As queries arrive in irregular patterns and each of them has different input and output lengths, new inference requests cannot be executed until the current batch is completed. As a solution to this issue, continuous batching techniques [13, 39] are widely adopted in existing LLM serving systems to improve inference throughput while minimizing delay. By batching requests at the iteration level and interleaving prefill and decoding stages, continuous batching can populate new requests into the batch being processed whenever an earlier request exits. For example, Figure 2(a) illustrates in the inference part how continuous batching handles three inference requests arriving at different timestamps. When a new request arrives, continuous batch processing will perform the prefill stage for it after the current decoding stage is finished. New requests and their KV cache tensors are then batched with unfinished requests for later decoding stages.

Nevertheless, more challenges arise when fine-tuning new LLMs and serving existing LLMs at the same time. Consider the following example in which a student is fine-tuning a large language model on a personal computer (PC) with a 24 GB GPU to help improve her thesis for an Ancient Greek course. For better results, she chose a 7B LLM because it is the largest LLM that fits in her PC, and PEFT because full parameter fine-tuning requires hundreds of GB of memory. After she starts the fine-tuning task, all other local LLM-based applications on the laptop will be unavailable because there are not enough resources to run their LLMs. Choosing a smaller LLM reduces the model's capabilities, while frequently terminating and starting fine-tuning and inference tasks reduces computational efficiency, as discussed below.

Switching between heterogenous LLM tasks mainly requires the following three steps: 1) clean the environment for the old task; 2) initialize the environment for the new task; and 3) load the LLM to be fine-tuned or served from CPU memory or storage to GPU memory. Due to the explosive growth of the LLM scale, the overhead of the third step has become very large and can no longer be avoided using GPU sharing technology [1, 2, 6]. Prior systems treat LLM training and inference as completely irrelevant processes. We profile two SoTA LLM systems (i.e., DeepSpeed [25] and Colossal-AI [18]) and show the breakdown of switching overhead in Table 1. It is clear that the overhead of the first two steps is tremendous.

Key ideas and motivations of this paper. In GPU clusters, throughput-intensive fine-tuning tasks and latency-sensitive inference tasks can be scheduled [12, 17, 32] to dedicated GPUs for training and inference separately. To achieve low-latency inference in resource-constrained environments, a naive solution is to switch to fine-tuning tasks when the system is idle. However, the system would constantly receive new inference requests, leaving little time for fine-tuning tasks. Another simple solution is to postpone all inference requests received when the system is fine-tuning new LLMs. While it can minimize fine-tuning time by incurring no switching overhead, it breaks the latency guarantee of inference. FineInfer sits in the middle of these two scenarios; with slightly deferred inferences that are still within service level agreements, we aim at a system that is able to greatly improve the latency of fine-tuning jobs.

3 FineInfer

In this section, we describe the design of FineInfer, a resource-efficient system optimized for heterogeneous LLM workloads. FineInfer builds on top of base model multiplexing and iteration-level switch to enable lightweight and fine-grained switching mechanism. At its core is deferred continuous batching, a new task-scheduling mechanism that improves fine-tuning throughput by slightly deferring inference requests without violating SLAs. LLMs fine-tuned through resource-efficient fine-tuning methods including PEFT can benefit from all designs of FineInfer. However, the iteration level switching and deferred continuous batching can be applied to other systems for general LLMs.

3.1 The Hybrid System

Existing systems are designed and aggressively optimized for either LLM training or inference. Instead, FineInfer adopts a hybrid system architecture that supports both fine-tuning and inference. This avoids coordinating two different systems on every switch, thereby mitigating task initialization and cleanup overhead. In addition, FineInfer incorporates optimizations including base model multiplexing and iteration-level switching, making the switching mechanism more lightweight and fine-grained.

Base model multiplexing. FineInfer extends the base model multiplexing [4, 8, 27] to minimize data movement overhead in task switch. When switching between fine-tuning and inference tasks, FineInfer only needs to swap the active adapters from those used for inference to the ones used for fine-tuning as opposed to the whole model. In other words, data movement between the GPU and CPU over the PCIe is reduced from tens of GBs to tens of MBs.

Iteration-level switching. FineInfer also extends the iteration-level scheduling [13, 39] in LLM inference to achieve fine-grained switching. If FineInfer is in fine-tuning mode and wants to serve newly arrived inference requests, it will

Algorithm 1 Deferred continuous batching.

Input: inference request queue Q_i, fine-tuning task queue Q_f, estimated completion time for a single fine-tuning iteration t_f

let B_i be the current batch of inference requests
let B_{new} be the batch of new inference requests
let B_f be the current batch of fine-tuning samples
while True **do**
  ▷ *FineInfer inference mode:* ◁
  $B_i \leftarrow \emptyset$
  while True **do**
    $B_{new} \leftarrow \emptyset$
    let t_c be the current timestamp
    for all $r \in Q$ **do**
      let t_r be the arrival timestamp of r
      let $d_r \leftarrow t_c - t_r$ be the deferred time of r
      let d_{rb} be the deferral bound of r
      if $d_r + t_f \geq d_{rb}$ **then**
        $B_{new} \leftarrow B_{new} \cup r$
    $Q \leftarrow Q \setminus B_{new}$
    if $B_{new} \neq \emptyset$ **then**
      let A_{new} be the adapters required by B_{new}
      $B_{new} \leftarrow$ inference_prefill(A_{new}, B_{new})
      $B_i \leftarrow B_i \cup B_{new}$
    if $B_i \neq \emptyset$ **then**
      let A_i be the adapters required by B_i
      $B_i \leftarrow$ inference_decode(A_i, B_i)
      $B_i \leftarrow B_i \setminus$ finished_requests(B_i)
    if $B_i = \emptyset$ **then**
      Break
  ▷ *FineInfer fine-tuning mode:* ◁
  $B_f \leftarrow$ get_first_batch(Q_f)
  $Q_f \leftarrow Q_f \setminus B_f$
  let A_f be the adapters required by B_f
  $A_f \leftarrow$ fine-tune(A_f, B_f) ▷ *Run only one iteration*

switch to inference mode upon completion of the current iteration. If already in inference mode, it batches the inference requests as continuous batching. In contrast to existing training systems that terminate the process only after completing the entire task or at least one epoch containing multiple iterations, FineInfer ensures that resources are promptly available for inference requests.

3.2 Deferred Continuous Batching

Deferred continuous batching aims to improve fine-tuning throughout while meeting latency SLA as illustrated in Figure 2. We describe in detail how and when it switches between different tasks in heterogeneous LLM workloads in Algorithm 1.

From fine-tuning to inference. When new inference requests with deferral bounds arrive, FineInfer does not process them immediately. Instead, it will be deferred until its deferred time is close to the deferral bound. Note that on a consumer GPU (e.g., the Nvidia 4090 GPU), the time required to process a batch of 8 inference requests is similar

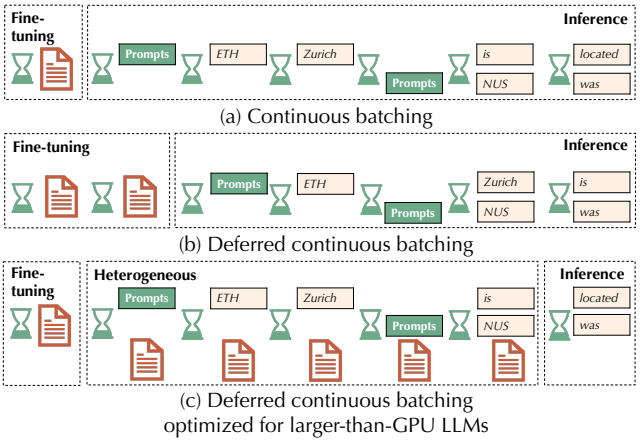

Figure 3. Offloading-based LLM execution under different scheduling strategies.

to the time required to process a single inference request. Therefore, deferred continuous batching exploits the batching opportunities for inference tasks and buys more time for fine-tuning tasks.

From inference to fine-tuning. If FineInfer is already in inference mode, it will not switch to fine-tuning mode until the completion of ongoing inference requests. We denote the arrival timestamp and the deferral bound of a new inference request as t_r and d_{rb}, the current timestamp as t_c, and the estimated time to complete a single fine-tuning iteration as t_f. If $t_c - t_r + t_f \geq d_{rb}$, FineInfer will batch new and ongoing requests starting from the next inference iteration. Otherwise, FineInfer will switch to fine-tuning mode after the completion of current inference requests.

When the deferral bound is small enough, the execution of deferred continuous batching is the same as the standard continuous batching, since the gap between adjacent requests is not even large enough for one fine-tuning iteration.

3.3 Deferred Continuous Batching for LLMs beyond GPU Memory

The explosive growth in the size of LLM and the limited GPU memory in resource-constrained environments have led to a series of offload-based LLM systems [5, 18, 26, 28, 30] that aggregate memory and computation from the GPU, CPU, and even disk to run LLMs. Pure deferred continuous batching is less effective in offload-based LLM executions because the bottleneck shifts to data movement between GPU and CPU. We propose to incorporate heterogeneous batching into the deferred continuous batching to optimize offload-based LLM executions.

Heterogeneous batching. Since the system has to run the LLMs multiple times to generate multiple tokens for an inference request, the model has to be loaded from CPU memory to GPU memory multiple times. For fine-tuning,

Algorithm 2 Deferred continuous batching optimized for larger-than-GPU LLMs.

Input: inference request queue Q_i, fine-tuning task queue Q_f, estimated completion time for a single inference iteration t_i, estimated completion time for a single fine-tuning iteration t_f

let $B_i \leftarrow \emptyset$ be the current batch of inference requests
let B_f be the current batch of fine-tuning samples
let F be the flag of whether to run fine-tuning
while True **do**
  let t_c be the current timestamp
  for all $r \in Q_i$ **do**
    let t_r be the arrival timestamp of r
    let $d_r \leftarrow t_c - t_r$ be the deferred time of r
    $B_i \leftarrow B_i \cup r$
  $Q_i \leftarrow Q_i \setminus B_i$
  $F \leftarrow$ True
  for all $r \in B_i$ **do**
    let d_{rb} be the deferral bound of r
    if $d_r + t_f - t_i \geq d_{rb}$ **then**
      $F \leftarrow$ False
  let A_i be the adapters required by B_i
  if F **then** ▷ *FineInfer heterogeneous mode*
    $B_f \leftarrow$ get_first_batch(Q_f)
    $Q_f \leftarrow Q_f \setminus B_f$
    let A_f be the adapters required by B_f
    loss , $B_i \leftarrow$ forward(A_i, B_i, A_f, B_f)
    gradients $\leftarrow$ backward(loss, A_f)
    $A_f \leftarrow$ update(A_f, gradients)
    for all $r \in B_i$ **do**
      $d_r \leftarrow d_r + t_f - t_i$
  else ▷ *FineInfer inference mode*
    $B_i \leftarrow$ forward(A_i, B_i)
  $B_i \leftarrow B_i \setminus$ finished_requests(B_i)

the system also requires loading the model for each batch of training samples, which presents a substantial opportunity to reduce data movement via batching. Therefore, FineInfer: 1) batches the input of one iteration of the inference requests and the input of a batch of training samples of the fine-tuning task; 2) loads each transformer layer of the model sequentially and runs forward passes of inference and fine-tuning tasks on this layer and corresponding adapters; and 3) runs backward pass of fine-tuning task. As shown in Figure 3, the combination of deferred continuous batching and heterogeneous batching significantly reduces data movement in GPU-CPU execution. Detailed pseudocode is included in Algorithm 2.

State and cache management. To improve computational efficiency, fine-tuning requires maintaining the optimizer state and activations, while inference requires maintaining a KV cache. Using the Llama2-7B model (FP16) as an example, if a fine-tuning task employs the rank-8 version of LoRA along with the AdamW optimizer (FP32), sets the batch size to 4, and aligns the length of each training sample to 256, an additional 3.14 GB of memory is needed to maintain optimizer states and activations. For an inference task with the same batch size, input sequence length of 224, and output sequence length of 32, an additional 0.5 GB of memory is required to maintain the KV cache.

However, when performing heterogeneous batching, existing systems will naively maintain and calculate the optimizer states, activations, and KV cache for both fine-tuning and inference tasks. Since the intermediate results are not released until the autoregressive generation process is completed, the memory consumption will be tens or hundreds of times that of fine-tuning or inference alone. Naively using heterogeneous batching in the fine-tuning and inference task examples above would result in an additional $32 * (3.14 + 0.5) = 116.48$ GB memory consumption.

To address these issues, we customize the computational graph of the fine-tuning task and the KV cache of the inference task to ensure that they only maintain and calculate intermediate results for the corresponding inputs. Meanwhile, FineInfer promptly deallocates the optimizer state and activations for fine-tuning between iterations, and the KV cache for inference between requests.

4 Evaluation

We now present a preliminary evaluation of FineInfer including comparisons against the state-of-the-art LLM systems Through experiments, we confirm the following:

- FineInfer effectively reduces the switching overhead between heterogeneous LLM tasks.
- In resource-constrained environments, FineInfer can complete fine-tuning tasks faster while meeting the latency guarantees of inference requests.
- When LLMs do not fit in GPU memory, FineInfer also outperforms strong state-of-the-art baselines.

4.1 Experimental Setup

We evaluate FineInfer on a dual-socket server with two 16-core AMD EPYC 7313 CPUs clocked at 3.0GHz, 256 GB of DRAM, and four Nvidia 3090 GPUs (24 GB). For all experiments, we ensure that only one GPU is enabled.

Models and datasets. We choose the Llama2 [34] models with 7B and 13B parameters as the base models and create PEFT adapter models from them and LoRA methods. In the fine-tuning task, we use the Alpaca [33] dataset and follow the default configuration for data preprocessing.

Workloads. We conducted experiments using synthetic workloads consisting of fine-tuning and inference, each lasting five minutes. For workloads of GPU-resident LLMs, new requests will arrive randomly within 2 seconds of the last request's arrival. For workloads of larger-than-GPU LLMs, new requests will arrive randomly within 1 minute of the last request's arrival. For all workloads, we assume that there are always unfinished or new tasks in the fine-tuning queue,

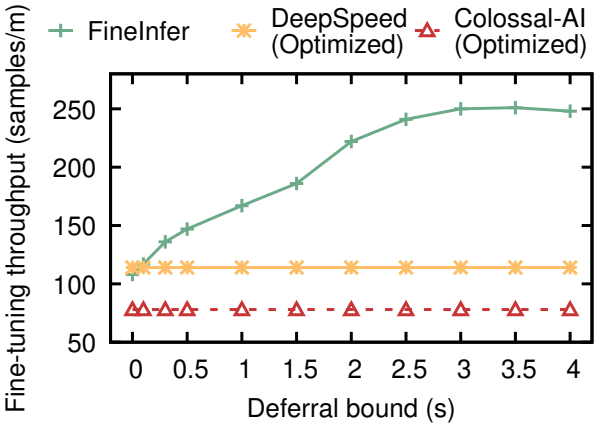

Figure 4. Experiment for LLMs fit in GPU memory. The benefits of deferred continuous batching increase with larger deferral bounds.

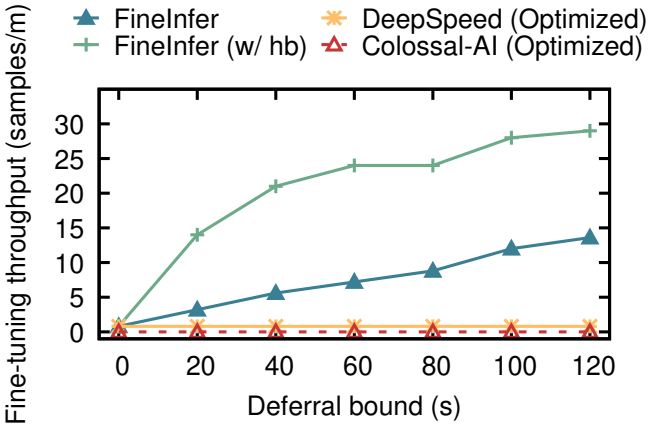

Figure 5. Experiment for larger-than-GPU LLMs. The combination of deferred continuous batching and heterogeneous batching boosts the improvement to 36x over the SoTA LLM systems by amortizing the data movement overhead with inference requests.

since each of them may take several minutes or tens of minutes.

Baselines. To the best of our knowledge, no other system has been optimized for concurrent LLM fine-tuning and inference. So we conduct experiments using variants of FineInfer and two popular open-source systems, DeepSpeed [25] and Colossal-AI [18]. As of February 2024, only these two systems fully support training, inference, and offloading. To ensure a fair comparison, the base model multiplexing and iteration-level switching are also implemented in the baselines when we verify the effectiveness of deferred continuous batching.

4.2 Switching Overhead

We run simple microbenchmarks consisting of either one fine-tuning iteration or one inference request to quantify the effectiveness of our design in reducing switching overhead. As Table 1 shows, FineInfer can reduce the data movement overhead from around 6 seconds to less than 0.1 seconds since base model multiplexing allows it to preserve the pre-trained model in GPU memory. At the same time, the hybrid system architecture avoids task initialization and cleanup overhead. The low end-to-end switching overhead of FineInfer makes running heterogeneous LLM workloads in resource-constrained workloads feasible.

4.3 GPU-Resident Performance

Our first end-to-end experiment evaluates the effectiveness of deferred continuous batching using Llama2-7B. Figure 4 plots how each variant behaves under inference requests with random arrival times and different deferral bounds. FineInfer outperforms all variants for small latency bounds and significantly minimizes fine-tuning time for larger latency bounds. The improvement is capped at 3.2x because when the deferral bound is large enough, inference cannot continue to increase its throughput by increasing the size of the batch.

4.4 Larger-than-GPU Performance

Next, we evaluate each variant with the Llama2-13B for GPU-CPU execution. As shown in Figure 5, FineInfer using only deferred continuous batching does not perform well in this case because the performance of this workload depends primarily on data movement overhead. By incorporating heterogeneous batching, FineInfer (w/ hb) performs 2.1x better than deferred continuous batching alone and can achieve throughput that is an order of magnitude higher (up to 29 samples per minute) than SoTA LLM systems (0.8 samples per minute).

5 Related Work

Our work builds upon rich literature on parameter-efficient fine-tuning, GPU scheduling and sharing, and offloading-based LLM systems.

Parameter-efficient fine-tuning. A substantial body of works [10, 15, 19, 21, 22] has been explored to reduce memory and compute demand of fine-tuning from an algorithmic perspective. They achieve this goal while providing promising statistical performance by updating a small set of parameters during fine-tuning. Although our evaluation focuses on the widely used LoRA, FineInfer can seamlessly integrate other PEFT techniques and lead to similar performance improvements.

GPU cluster scheduling. Scheduling strategies [12, 17, 23, 40] have been extensively explored to maximize resource utilization of deep learning (DL) workloads on GPU clusters. The above approach assigns DL jobs to appropriate machines

or GPUs based on compute, memory, and bandwidth requirements, and is therefore not suitable for resource-constrained environments. Exploring scheduling strategies to co-locate DL jobs using the same pre-trained model in a GPU cluster, and then applying our work, would be an interesting future task.

GPU sharing. GPU sharing can be categorized into spatial sharing and temporal sharing. Spatial sharing [1, 2, 31] allows multiple processes to run on different regions of the same GPU simultaneously, eliminating task switch overheads. However, using spatial sharing requires the system to hold at least two copies of models, one for inference and one for fine-tuning. Compared to our solution, spatial sharing must sacrifice statistical performance because it can only use smaller models to meet memory constraints.

Temporal sharing [6] temporally multiplexing the GPU by context-switching between multiple jobs to improve utilization. Unlike spatial sharing, it dedicates all resources of a GPU to a single job for a time slice. Compared to time-sharing solutions, heterogeneous batching allows FineInfer to serve PEFT and inference simultaneously.

Offloading-based LLM systems. FlexGen [28] optimizes throughput-oriented LLM inference for latency-insensitive scenarios via zig-zag block schedule. PowerInfer [30] and LLM-in-a-flash [30] leverage the activation sparsity of ReLU-based LLMs to reduce CPU-GPU data movement in LLM inference, but these methods require retraining after modifying the model architecture. DeepSpeed [26] and Colossal-AI [18] both provide offloading solutions in their training systems and inference systems. They can achieve advanced performance in homogeneous training or inference tasks but are not optimal for heterogeneous LLM workloads due to the overhead of coordinating two different systems.

6 Conclusion and Future Work

We introduce FineInfer, the first system designed for concurrent LLM fine-tuning and inference in resource-constrained environments. FineInfer uses a combination of new and extended existing techniques, including (1) base model multiplexing and iteration-level scheduling to minimize switching overhead, (2) deferred continuous batching to orchestrate resource-efficient fine-tuning and inference, and (3) heterogeneous batching to reduce data movement. Evaluation results show that on an Nvidia 4090 GPU, FineInfer can reduce fine-tuning time by up to 3x for GPU-resident LLMs and 36x for larger-than-GPU LLMs compared to existing systems under different workloads while ensuring the latency guarantees of inference requests.

In the future, we aim to include more resource-constrained devices (e.g., personal computers and mobile phones) and metrics (e.g., SLAs) in the evaluation. Additionally, we will use statistical modeling [9] or real-world LLM job traces [11, 16, 36] to better generate workloads.

Acknowledgments

We thank Lequn Chen for the valuable discussions. We also thank the anonymous reviewers for their constructive feedback.

References

[1] 2013. NVIDIA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html

[2] 2020. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[3] 2023. NeurIPS Large Language Model Efficiency Challenge: 1 LLM + 1GPU + 1Day. https://llm-efficiency-challenge.github.io/

[4] 2023. Parameter-Efficient Fine-Tuning. https://github.com/huggingface/peft

[5] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *CoRR* abs/2312.11514 (2023).

[6] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 499–514.

[7] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ B. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the Opportunities and Risks of Foundation Models. *CoRR* abs/2108.07258 (2021). https://arxiv.org/abs/2108.07258

[8] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023. Punica: Multi-Tenant LoRA Serving. *CoRR* abs/2310.18547 (2023).

[9] Robert Gallager. 2011. Chapter 2: Poisson Processes. In *6.262 Discrete Stochastic Processes*. Massachusetts Institute of Technology.

[10] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[11] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. 2024. Characterization of large language model development in the datacenter. (2024).

[12] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Lucid: A Non-intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*. ACM, 457–472.

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

23-26, 2023. ACM, 611–626.

[14] Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomás Kociský, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. 2021. Mind the Gap: Assessing Temporal Generalization in Neural Language Models. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 29348–29363.

[15] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 3045–3059.

[16] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently adapting learned cardinality estimators to data and workload drifts. In *Proceedings of the 2022 International Conference on Management of Data*. 1920–1933.

[17] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2023. Lyra: Elastic Scheduling for Deep Learning Clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*. ACM, 835–850.

[18] 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, ICPP 2023, Salt Lake City, UT, USA, August 7-10, 2023*. ACM, 766–775.

[19] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 4582–4597.

[20] Yikang Li, Nan Duan, Bolei Zhou, Xiao Chu, Wanli Ouyang, Xiaogang Wang, and Ming Zhou. 2018. Visual question generation as dual task of visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6116–6124.

[21] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[22] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 61–68.

[23] Yao Lu, Song Bian, Lequn Chen, Yongjun He, Yulong Hui, Matthew Lentz, Beibin Li, Fei Liu, Jialin Li, Qi Liu, Rui Liu, Xiaoxuan Liu, Lin Ma, Kexin Rong, Jianguo Wang, Yingjun Wu, Yongji Wu, Huanchen Zhang, Minjia Zhang, Qizhen Zhang, Tianyi Zhou, and Danyang Zhuo. 2024. Computing in the Era of Large Generative Models: From Cloud-Native to AI-Native. *CoRR* abs/2401.12230 (2024).

[24] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems. *CoRR* abs/2312.15234 (2023). https://doi.org/10.48550/ARXIV.2312.15234 arXiv:2312.15234

[25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 3505–3506.

[26] 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 2021, July 14-16, 2021*. USENIX Association, 551–564.

[27] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2023. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. *CoRR* abs/2311.03285 (2023).

[28] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 31094–31116.

[29] 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. *CoRR* abs/1909.08053 (2019).

[30] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. *CoRR* abs/2312.12456 (2023).

[31] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM.

[32] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. 2023. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*. ACM, 642–657.

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

[34] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023).

[35] Lukas Tuggener, Pascal Sager, Yassine Taoudi-Benchekroun, Benjamin F Grewe, and Thilo Stadelmann. 2024. So you want your private LLM at home?: a survey and benchmark of methods for efficient GPTs. In *11th IEEE Swiss Conference on Data Science (SDS), Zurich, Switzerland, 30-31 May 2024*. ZHAW Zürcher Hochschule für Angewandte Wissenschaften. https://digitalcollection.zhaw.ch/bitstream/11475/30279/3/2024_Tuggener-etal_Survey-and-benchmark-of-methods-for-efficient-GPTs_SDS.pdf

[36] Yuxin Wang, Yuhan Chen, Zeyu Li, Zhenheng Tang, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2024. Towards Efficient and Reliable LLM Serving: A Real-World Workload Study. *CoRR* abs/2401.17644 (2024).

[37] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and optimizing machine learning workflows on heterogeneous infrastructures. *arXiv preprint arXiv:2205.04713* (2022).

[38] Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, Qiyang Zhang, Zhenyan Lu, Li Zhang, Shangguang Wang, Yuanchun Li, Yunxin Liu, Xin Jin, and Xuanzhe Liu. 2024. A Survey of Resource-efficient LLM and Multimodal Foundation Models. *CoRR* abs/2401.08092 (2024).

[39] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*. USENIX Association, 521–538.

[40] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Ré, and Ce Zhang. 2022. Decentralized Training of Foundation Models in Heterogeneous Environments. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Received 26 February 2024; accepted 22 March 2024