Loquetier: A Virtualized Multi-LoRA Framework for Unified LLM Fine-tuning and Serving

Yuchen Zhang[1] **Hanyue Du**[1] **Chun Cao**[1] **Jingwei Xu**[1]*
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
`{zfirozen,dhy}@smail.nju.edu.cn, {caochun,jingweix}@nju.edu.cn`

Abstract

Low-Rank Adaptation (LoRA) has become a widely adopted parameter-efficient fine-tuning (PEFT) technique for adapting large language models (LLMs) to downstream tasks. While prior work has explored strategies for integrating LLM training and serving, there still remains a gap in unifying fine-tuning and inference for LoRA-based models. We present **Loquetier**, a virtualized multi-LoRA framework that seamlessly integrates LoRA fine-tuning and serving within a single runtime. Loquetier introduces two key components: (1) a *Virtualized Module* that isolates PEFT-based modifications and supports multiple adapters on a shared base model, and (2) an optimized computation flow with a kernel design that merges fine-tuning and inference paths in forward propagation, enabling efficient batching and minimizing kernel invocation overhead. Extensive experiments across three task settings show that Loquetier consistently outperforms existing baselines in both performance and flexibility, achieving up to $3.0\times$ the throughput of the state-of-the-art co-serving system on inference-only tasks and $46.4\times$ higher SLO attainment than PEFT on unified fine-tuning and inference tasks. The implementation of Loquetier is publicly available at `https://github.com/NJUDeepEngine/Loquetier`.

1 Introduction

Large Language Models (LLMs) built on stacked transformer blocks [Vaswani et al., 2017] have achieved remarkable success across a wide range of text generation tasks. This success has driven the development of ever-larger models, such as the LlaMA series [Touvron et al., 2023, Grattafiori et al., 2024] and the Qwen family [Bai et al., 2023, Yang et al., 2024]. However, the rapid growth in model size has introduced prohibitive costs. For example, LlaMA 3 contains 405B parameters, and DeepSeek-V3 [Bi et al., 2024] scales to 671B. Their computational and memory requirements of full-parameter training now represent a major bottleneck, restricting both the scalability and accessibility of LLM development.

Parameter-efficient fine-tuning (PEFT) has emerged as a practical solution to these challenges. By reducing the number of trainable parameters while retaining the effectiveness of full-model fine-tuning, PEFT offers a balance between efficiency and adaptability [Ding et al., 2023a]. Recent studies have evaluated PEFT across diverse applications and theoretical dimensions [Balne et al., 2024, Xu et al., 2023a], and have surveyed its underlying mechanisms and practical benefits [Han et al., 2024, Fu et al., 2023]. Notice that PEFT approaches such as Prefix Tuning [Li and Liang, 2021] and Prompt Tuning [Lester et al., 2021] have demonstrated strong adaptability by optimizing small task-specific vectors. Moreover, PEFT has shown competitive or superior performance in low-resource settings, including zero- and few-shot learning [Liu et al., 2022, Hu et al., 2023].

*Corresponding author

Table 1: Comparison on different LoRA tasks between Loquetier, PEFT and FlexLLM

	Inference		Finetune		Finetune & Inference	
Framework or System	Single	Multi	Single	Multi	Single	Multi
Loquetier	✓	✓	✓	✓	✓	✓
PEFT	✓	✓	✓	×	✓	×
S-LoRA+PEFT	✓	✓	✓	×	✓	×
FlexLLM	✓	△[3]	✓[4]	×[4]	×	×

Among PEFT approaches, *Low-Rank Adaptation* (LoRA) [Hu et al., 2022] has become particularly prominent, offering scalable and effective fine-tuning across diverse LLM tasks. Numerous extensions have been proposed to enhance its flexibility, including LoHa [Hyeon-Woo et al., 2021], VeRA [Kopiczko et al., 2023], and LoKr [Yeh et al., 2023]. LoRA has also shown strong potential for personalization, powering systems in recommendation [Kong et al., 2024, Zhu et al., 2024], and user-centered content generation [Zhang et al., 2024, Wu et al., 2024a]. Its modularity makes it well-suited for large-scale serving systems where real-time customization is critical.

In practice, systems that serve LoRA often need to support fine-tuning adapters while simultaneously deploying them for inference across diverse tasks. However, no existing framework can seamlessly unify these two capabilities, resulting major obstacles for scaling LoRA into production. Jointly fine-tuning and serving multiple adapters requires minimizing memory and computation overhead while efficiently handling heterogeneous workloads. Prior efforts have mainly optimized the base LLM, such as through KVCache improvements or inference pipeline parallelization [Li et al., 2024a], with only limited advances in multi-LoRA inference [Chen et al., 2024, Sheng et al., 2023] or integrated fine-tuning and inference [Miao et al., 2024]. However, these approaches still face critical challenges: adapters are often fused into monolithic instances for efficiency, thus they cannot be dynamically loaded or unloaded; decoding efficiency degrades significantly when fine-tuning and inference run concurrently; and task switching typically requires halting the current job before starting another, causing downtime, bandwidth overhead, and wasted resources. As a result, existing frameworks remain inadequate for large-scale, production-ready LoRA applications.

In this paper, we introduce **Loquetier**[2], a unified virtualization framework that integrates fine-tuning and serving of LLMs with LoRA-based PEFT. Loquetier provides a streamlined computation flow that handles both fine-tuning and inference requests within a shared runtime, using kernel-level optimizations to reduce memory overhead and execution latency. Specifically, Loquetier introduces the *Segmented Multi-LoRA Multiplication (SMLM)* kernel, which enables mixed-task execution by distinguishing forward-pass behaviors for fine-tuning, evaluation, prefilling, and decoding. To support multiple concurrent LoRA adapters without modifying the base model, Loquetier further incorporates a *Virtualized Module* abstraction that dynamically injects adapter logic while preserving compatibility and isolation across tasks and devices. This design enables seamless co-serving of heterogeneous LoRA configurations and supports instance-to-instance migration of fine-tuning jobs without kernel restarts or memory duplication. The main contributions are as follows:

- We design an SMLM kernel and an unified computation flow that efficiently supports fine-tuning and inference with multiple LoRA adapters on a shared base model.
- We propose a modular virtualization mechanism that isolates PEFT-based modifications from the base model, enabling flexible instance-level migration and seamless adapter management.
- We develop Loquetier to unify LoRA fine-tuning and serving. Extensive experiments show that Loquetier outperforms existing systems across diverse scenarios and enables unified practical fine-tuning and serving configurations previously unsupported.

[2]The name *Loquetier* is a synthesis of "LoRA" and "coquetier", reflecting our design philosophy: the base model serves as the foundational spirit, while LoRA modules act as adjunct ingredients-spirits, juices, and syrups-mixed in for task-specific customization.

[3]FlexLLM cycles through loading LoRA models during multi-LoRA inference, disregarding the maximum number of resident LoRAs set, which makes its multi-LoRA inference efficiency practically unusable.

[4]The backward procedure of FlexLLM triggered an error originating from an unsupported operation in its gradient computation logic. We describe our solution in the Appendix B.

2 Related Work

LLM inference optimization via KV cache. Key-value (KV) caching is a core technique for accelerating LLM inference by avoiding redundant computation and memory transfers [Pope et al., 2023]. Recent work improves cache efficiency and reduces GPU memory overhead through tailored management strategies. Prompt Cache [Gim et al., 2024] reuses prompt embeddings to reduce duplication. vLLM [Kwon et al., 2023] and vAttention [Prabhu et al., 2025] address fragmentation using paging and virtual memory, respectively. Infinite-LLM [Lin et al., 2024] enables dynamic sharing of cache segments between host and GPU. LoongServe [Wu et al., 2024b] enhances long-context serving by balancing prefilling and decoding workloads.

LLM inference parallelism. A large amount of work improves LLM inference throughput by optimizing batch scheduling and pipeline execution. Response Length Perception [Zheng et al., 2023] and S^3 [Jin et al., 2023] predict output lengths to batch similar requests, with S^3 further refining its predictions over time. Orca [Yu et al., 2022] uses token-level continuous batching, while DeepSpeed-Fastgen [Holmes et al., 2024] adjusts request lengths for better GPU utilization. Sarathi-Serve [Agrawal et al., 2024] co-schedules prefilling and decoding tasks, whereas TetriInfer [Hu et al., 2024], Splitwise [Patel et al., 2024], and DistServe [Zhong et al., 2024] decouple these phases across threads, machines, or clusters to improve parallel efficiency. In kernel, FlashDecoding++ [Hong et al., 2024] overlaps computation with data transfer to hide latency. FlashAttention [Dao, 2023, Shah et al., 2024] uses warp specialization and asynchronous execution to maximize attention-layer throughput.

Efficient multi-LoRA inference and unified fine-tuning-inference systems. FlashInfer [Ye et al., 2025] reduces redundant storage and optimizes GPU memory access for efficient multi-LoRA inference. Cutlass [Thakkar et al., 2023], a high-performance CUDA library for GEMM operations, serves as the kernel foundation for Punica [Chen et al., 2024], which combines with FlashInfer to support scalable LoRA serving. Built on this stack, S-LoRA [Sheng et al., 2023] further improves GPU memory utilization through dynamic host-device memory transfers. In parallel, FlexLLM [Miao et al., 2024] explores unified token-level computation for co-serving inference and PEFT fine-tuning, though scalability remains limited.

LoRA optimization and variants. There are lots of extensions that have been proposed to enhance the flexibility, efficiency, and adaptability of LoRA. AdaLoRA [Zhang et al., 2023a] introduces singular value decomposition for dynamic pruning, while IncreLoRA [Zhang et al., 2023b] allocates parameters based on module importance. SoRA [Ding et al., 2023b] adaptively adjusts rank via gated weight control during training. DiffoRA [Jiang et al., 2025] selects modules to finetune using a Differential Adaptation Matrix (DAM). To improve expressiveness and stability, DoRA [Liu et al., 2024] decomposes weights into magnitude and direction. VB-LoRA [Li et al., 2024b] reduces redundancy by sharing global vector banks across modules. LoRA-XS [Bałazy et al., 2024] compresses storage with minimal $r \times r$ matrices, and QA-LoRA [Xu et al., 2023b] integrates quantization and grouping to increase representational flexibility.

3 Loquetier Framework

In this section, we describe the overall architecture of Loquetier, followed by details of its unified computation flow and the proposed LoRA model virtualization.

3.1 Framework

As illustrated in Figure 1, Loquetier framework consists of two core components: (1) a model library centered around the *Virtualized Module* for isolating PEFT modifications, and (2) a redesigned kernel and computation flow that jointly support fine-tuning and inference workloads. To enable concurrent execution, Loquetier integrates a context management system that coordinates the runtime scheduling of heterogeneous tasks.

When loading a base model into CPU or GPU memory, Loquetier instantiates multiple *virtual models*, each acting as an isolated container for a specific PEFT configuration. These virtual models are bound to distinct adapters, enabling independent and concurrent execution. For LoRA-based adapters, we introduce the `MixedLoraModel` class to support fine-tuning. Based on our computation flow (see Section 3.3), each `MixedLoraModel` efficiently fine-tunes its associated LoRA adapter within

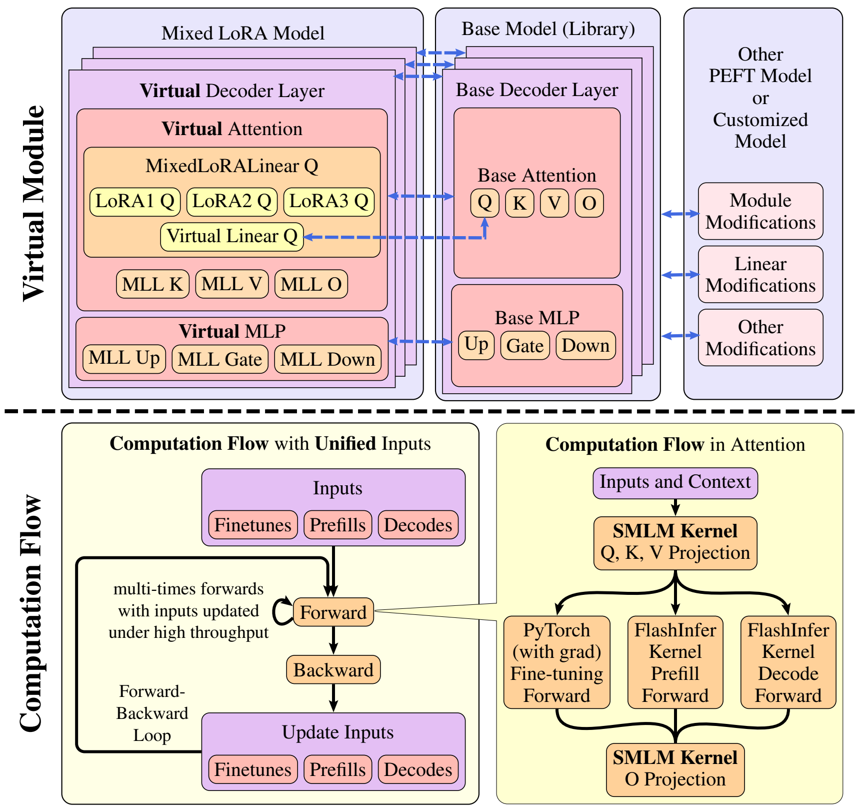

Figure 1: The framework diagram of Loquetier.

its own container from dedicated trainers, while other virtual models can simultaneously remain in inference mode.

Loquetier's modular design is compatible with most existing LLM inference optimizations and training strategies. It fully supports architectures that leverage FlashInfer [Ye et al., 2025] kernel and remains extensible to those that do not, requiring only localized modification to the inference logic within the computation flow. Section 3.2 further details how the Virtualized Module enables base model sharing across diverse PEFT methods beyond LoRA.

3.2 Virtualized module

The mixing of different LoRA or other PEFT methods in the same model object makes model configurations chaotic and difficult to handle. Moreover, dynamic model loading and unloading should be supported in order to apply the fine-tuned and up-to-date LoRA models quickly.

We propose the Virtualized Module that provides methods and data proxies to foundation modules to solve the above problems. Applying the Virtualized Module to the base model is extremely low-cost, with no additional GPU memory overhead and provides independent model instances in an intuitive way. For each module type, the Virtualized Module creates the corresponding virtual module at runtime to provide the correct proxy methods. The Virtualized Module prepares additional properties and methods for Linear and Model to accommodate the architecture of Transformers and PEFT.

Since each virtual module class is created at runtime, which means none of these virtual modules can be shared, nor can a process that owns any virtual module create child processes from fork or spawn method. Furthermore, any deep copying behavior that includes a virtual module will cause the base module linked to it to be copied, thus invalidating base module sharing. For this reason, we provide a non-local class definition for deep copying, serialization, and deserialization. By voiding the containing Virtualized Module, LoRA models and other PEFT models loaded onto the Virtualized

Algorithm 1 Computation flow control in attention layer of Loquetier

```
Input: hidden states matrix $\mathbf{X}$ with shape $[S, H]$, list of fine-tuning inputs batch-sequence information tuples $\mathbf{F}$, list of prefilling and evaluation inputs sequence lengths $\mathbf{P}$, decoding inputs count $\mathbf{D}$.
Output: attention outputs matrix $\mathbf{O}$.
  $\mathbf{Q} = Q_{proj}(\mathbf{X})$; $\mathbf{K} = K_{proj}(\mathbf{X})$; $\mathbf{V} = V_{proj}(\mathbf{X})$;
  $\mathbf{Os} = []$;
  if $len(\mathbf{F}) > 0$ then
      Extract $\mathbf{Q_f}, \mathbf{K_f}, \mathbf{V_f}$ from $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ based on $\mathbf{F}$;
      Compute $\mathbf{O_f}$ through the standard forward implementation;
      $\mathbf{O_f}$ is appended to $\mathbf{Os}$;
  end if
  if $len(\mathbf{P}) > 0$ then
      Compute offset of prefills $\mathbf{Offset_p}$ based on $\mathbf{F}$;
      Extract $\mathbf{Q_p}, \mathbf{K_p}, \mathbf{V_p}$ from $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ based on $\mathbf{P}$ and $\mathbf{Offset_p}$;
      Initialize KVCache for prefills;
      Compute $\mathbf{O_p}$ through the FlashInfer forward implementation;
      $\mathbf{O_p}$ is appended to $\mathbf{Os}$;
  end if
  if $\mathbf{D} > 0$ then
      Compute offset of decodes $\mathbf{Offset_d}$ based on $\mathbf{F}$ and $\mathbf{P}$;
      Extract $\mathbf{Q_d}, \mathbf{K_d}, \mathbf{V_d}$ from $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ based on $\mathbf{Offset_d}$;
      Append KVCache for decodes;
      Compute $\mathbf{O_d}$ through the standard forward implementation;
      $\mathbf{O_d}$ is appended to $\mathbf{Os}$;
  end if
  Concatenate $\mathbf{Os}$ into one tensor as $\mathbf{O}$;
  $\mathbf{O} = O_{proj}(\mathbf{O})$;
  return $\mathbf{O}$;
```

Module can be migrated to other GPU devices after deep-copying and used after unvoiding based on instances of the new Virtual Model.

Based on the above design, the Virtualized Module is compatible with all PEFT methods, as well as any custom model modification methods that do not modify the underlying model's own data, such as weights and module configurations. For scenarios where the target module overwrites the base module with new data, the module design needs to be normalized so that the target module runs the new forward method based on its own data and the base module's data, rather than running the base module's forward method after using destructive modification methods.

3.3 Unified computation flow management and SMLM kernel

For multiple LoRA adapters within the same linear layer, traditional methods typically process the computation sequentially, computing the output for one LoRA at a time and iterating through all adapters. This approach significantly slows computation. Since different LoRAs within the same layer usually share identical or similar shapes, it is feasible to compute all outputs in a single kernel call. Punica [Chen et al., 2024] leverages this property by implementing a foundational algorithm on top of the Cutlass GEMM (General Matrix Multiplication) library [Thakkar et al., 2023], enabling simultaneous processing of multiple input-LoRA pairs for efficient multi-adapter computation.

However, the original Punica kernel design is incompatible with fine-tuning, as it statically concatenates LoRA weights from the same module across different layers. This rigid coupling limits architectural flexibility and prevents selective application of LoRA to specific layers, which is particularly problematic in training scenarios where layerwise heterogeneity is common. This requires every layer in fine-tuning tasks to adopt the exact same configuration, even when certain linear modules do not actually require LoRA. Meanwhile, this design incurs substantial overhead in tracking and memory management, especially when operating on very large tensors during training. For inference tasks,

Algorithm 2 Computation flow control in causal LM of Loquetier

```
Input: LM inputs $\mathbf{X}$, list of fine-tuning inputs batch-sequence information tuples $\mathbf{F}$, list of evaluation
    inputs sequence lengths $\mathbf{E}$, Labels of fine-tuning and evaluation inputs $\mathbf{Labels}$, Accumulation
    steps of fine-tuning and evaluation inputs $\mathbf{A}$.
Output: list of losses $\mathbf{Loss}$, logits $\mathbf{Logits}$.
  Compute $\mathbf{Logits}$ by forward propagation;
  $\mathbf{Loss} = []$
  for (batch-sequence $(\mathbf{B}, \mathbf{S})$, accumulation step $\mathbf{A_{FE}}$ in $(\mathbf{F}, \mathbf{E}), \mathbf{A}$) do
      Extract $\mathbf{Logits_{FE}}$ and $\mathbf{Labels_{FE}}$ from $\mathbf{Logits}$ and $\mathbf{Labels}$;
      Shift $\mathbf{Logits_{FE}}$ and $\mathbf{Labels_{FE}}$;
      Compute $\mathbf{Loss_{FE}}$ of $\mathbf{Logits_{FE}}$ and $\mathbf{Lables_{FE}}$ from the given loss function;
      $\mathbf{Loss_A} = \mathbf{Loss_{FE}} / \mathbf{A_{FE}}$;
      $\mathbf{Loss_A}$ is appended to $\mathbf{Loss}$;
  end for
  return $\mathbf{Loss}, \mathbf{Logits}$;
```

it also eliminates the possibility of rapidly swapping LoRAs during runtime. Thus, the computation process must be halted before replacement, and the required LoRA must be re-spliced together.

To overcome these limitations, we adapt the Punica kernel to process LoRA weights one linear layer at a time. This decoupling removes the need to regenerate model files via weight transformation before execution. For static scaling factors in LoRA, we apply the scale directly to the weight tensor at `MixedLoraModel` instantiation to reduce runtime overhead. When dynamic scaling is required, it is applied on a per-request basis during the forward pass. Loquetier then executes forward computation across all active requests in a unified manner, supporting four types of requests: fine-tuning (training), evaluation, prefilling, and decoding. Evaluation requests are structurally similar to prefilling but compute a loss over labels and execute only a single generation pass, while prefilling requests omit the loss computation and transition into decoding after the initial pass.

For the backward pass, since FlashInfer does not support gradient computation, Loquetier falls back to the standard forward implementation backed by PyTorch's Autograd when handling fine-tuning requests. This enables full gradient computation through efficient C++-based differentiation. For inference-only requests, Loquetier instead leverages FlashInfer's batch-prefill and batch-decode kernels to maximize throughput and memory efficiency.

The forward computation flow for the attention layer is summarized in Algorithm 1. First, the **Q**, **K**, and **V** projections are computed jointly for all incoming requests. Attention outputs are then computed independently for each request type, concatenated, and passed through a shared output projection **O**. Algorithm 2 outlines the forward method for the causal language model. For each incoming request, Loquetier computes output logits and, when applicable, the loss with respect to provided labels. Fine-tuning and evaluation requests include ground truth labels and thus return both logits and loss values, while prefilling and decoding requests return only logits. Loquetier enables joint forward pass of fine-tuning and evaluation requests within the same batch. Because the losses are tracked separately in Loquetier, this separation allows distinct gradient accumulation strategies for different fine-tuning tasks in parallel, without cross-interference. By summing losses across all fine-tuning requests, Loquetier produces a shared backward pass, enabling gradients from multiple fine-tuning jobs to be computed efficiently in a single backpropagation step.

We further extend the Transformers' Trainer [Wolf et al., 2020] to support an interruptible fine-tuning process. Multiple trainers can now share the same computation flow in Loquetier, performing unified forward and backward propagation for fine-tuning different LoRA adapters concurrently. To ensure that each trainer only updates its corresponding parameters, we introduce `MixedLoRAModelForTrainer`, which applies parameter masking on top of a shared `MixedLoraModel` instance to achieve isolation.

4 Experiments

The Loquetier framework expects to enable fine-tuning and reasoning across multiple LoRA models. Based on the objective, we design three experiments to evaluate the performance of Loquetier in

Table 2: Comparison on model loading between Loquetier, PEFT, S-LoRA and FlexLLM. Metrics include time to load (Time) and additional storage footprint (Storage).

	Base Model		LoRA Model		Total	
Framework or System	Time	Storage	Time	Storage	Time	Storage
Loquetier	2.927 s	0 B	2.409 s	0 B	5.336 s	0 B
PEFT	2.877 s	0 B	1.914 s	0 B	4.791 s	0 B
S-LoRA	33.037 s	0 B	0.948 s	0 B	33.985 s	0 B
FlexLLM	37.933 s	14.96 GB	0.924 s	40.04 MB	38.857 s	15.00 GB

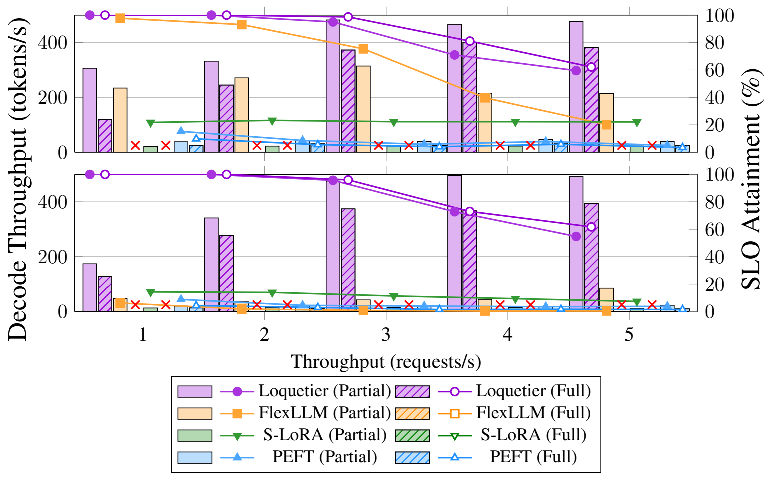

Figure 2: Comparison of the performance of Loquetier, FlexLLM, S-LoRA and PEFT in inference tasks. The upper is single LoRA model inference and the lower part is multiple LoRA model inference. Partial means that only 3 modules are enabled for FlexLLM including up, gate, and down. For detailed information on S-LoRA, please refer to the Appendix E. Full means that all 7 modules are enabled, including q, k, v, o, up, gate, and down. $\times$ indicates that the results were not obtained: FlexLLM does not support enabling LoRA modules for linear layers other than up, gate, and down; FlexLLM cycles through loading LoRA models during multi-LoRA inference.

these scenarios: inference-only, fine-tuning-only, and unified fine-tuning and inference. In addition, we perform two experiments to evaluate the performance of Loquetier in simulated real-world environments: a shorter, approximate simulation for rapidly testing different load conditions across different times of the day, and a 120-minute precise simulation using data extracted from real-world scenarios for a more demanding and realistic stress test. To further validate the effectiveness of our design, we conduct a micro-experiment to evaluate the efficiency of the model loading process.

4.1 Evaluation settings

Baselines. FlexLLM is an advanced co-serving system for LLM serving and parameter-efficient fine-tuning. We deployed docker images as its runtime environment following the guidelines. S-LoRA is designed specifically for large-scale LoRA inference. Therefore, we combine S-LoRA with PEFT as another baseline, in which PEFT handles the fine-tuning task. We use HuggingFace Transformers with PEFT as the most basic baseline.

Models. We use the Llama3-8B model as the base model. The LoRA adapter is obtained by training on the Alpaca dataset. For the fine-tuning task, we use the same LoRA configuration as the inference LoRA adapter and initialize the weights from the Gaussian distribution.

Datasets. We use the ShareGPT dataset as input for the inference task, and the Alpaca and GSM8K datasets as input for the fine-tuning task. BurstGPT [Wang et al., 2025] is an LLM service workload dataset comprising over ten million traces collected from Azure OpenAI GPT services. Data spanning more than 60 days is segmented into 20-minute slices according to its provided partitioning scheme.

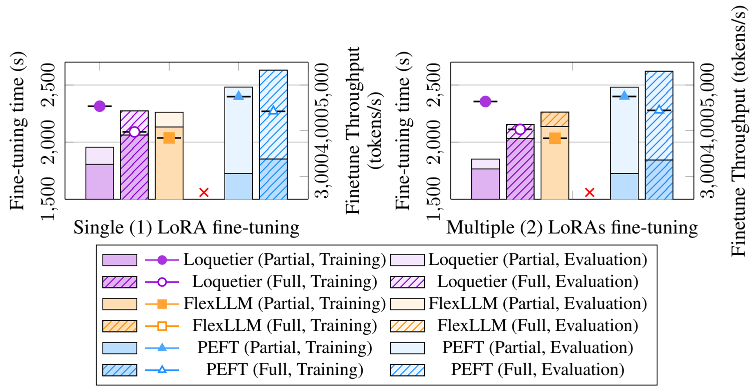

Figure 3: Comparison of the performance of Loquetier, FlexLLM and PEFT in fine-tuning tasks. The meanings of Partial and Full are the same as in Figure 2. $\times$ indicates that the results were not obtained: FlexLLM does not support backward propagation computations for modules other than up, gate and down. PEFT can only finetune one LoRA adapter at a time, so its time cost is cumulative.

Hardware. We test the inference-only tasks on a server with NVIDIA A6000 48G GPUs. We test the fine-tuning-only tasks and the unified fine-tuning and inference tasks on servers with NVIDIA H800 80G GPUs. Each test process has at least 128G of host memory available.

Metrics and Tasks. The metrics used for the evaluation are listed in Appendix C. The inference-only tasks run inference tasks at different request arrival rates where inference needs to be as fast as possible to achieve the Service Level Objective (SLO). The fine-tuning-only tasks need to run the training task for a specified number of epochs to measure its efficiency in processing tokens. The unified fine-tuning and inference tasks need to run the training task along with the inference task, weighing the efficiency of fine-tuning tokens against the SLO of the inference request. The real-world workload simulation experiment samples six time periods from the BurstGPT dataset, comprising one low-load, two medium-load, and three high-load intervals. Each slice was categorized into one of three load tiers based on its request rate: low-load periods for average RPS < 1; medium-load for 1~1.75; and high-load for > 1.75 (which may include transient spikes exceeding RPS 10). Here, RPS denotes requests per second. The detailed configurations can be viewed in Appendix D.

4.2 Evaluation results

Model Loading. The loading speed and additional loading storage overhead are shown in Table 2. Compared to PEFT, Loquetier requires creating Virtualized Modules and applying scaling to each LoRA linear when loading LoRA models, resulting in a slight slowdown in this part of the loading. FlexLLM needs to transform and cache the model weights, which leads to a significant additional storage footprint. Even with cached transformed models, FlexLLM's loading is still very slow due to the need to read small weight files.

Inference. Figure 2 shows the test results of the inference task. Loquetier maintains the highest SLO attainment at different request arrival rates. As the request rate increases, Loquetier's decoding speed gradually increases. Until at 3 RPS, the decoding speed no longer increases, indicating that the GPU memory access bottleneck has been hit. As the request rate continues to increase, some requests begin to fail to reach their SLOs due to the inability to achieve faster inference on the current GPU.

FlexLLM's maximum decoding speed is lower than Loquetier's, causing its SLO attainment rate to start dropping earlier and fall off a cliff at higher request arrival rates. In addition, in conjunction with the findings in Section 4.2, FlexLLM's lazy loading mechanism prevents it from handling some of the earliest arriving requests under SLO. Forcing early loading of the model weights improves SLO attainment to or near 100% at 1-2 RPS, but the improvement is very limited for higher request arrival rates due to the limitations of its highest decoding speed. FlexLLM is unable to apply LoRA to all 7 modules, causing it to fail under the corresponding experiments. When loading multiple LoRA models, FlexLLM is trapped in a dead loop for more than 10 minutes, missing SLOs for all requests. After a longer wait, FlexLLM cannot get out of the trap still, and therefore is marked as a failure.

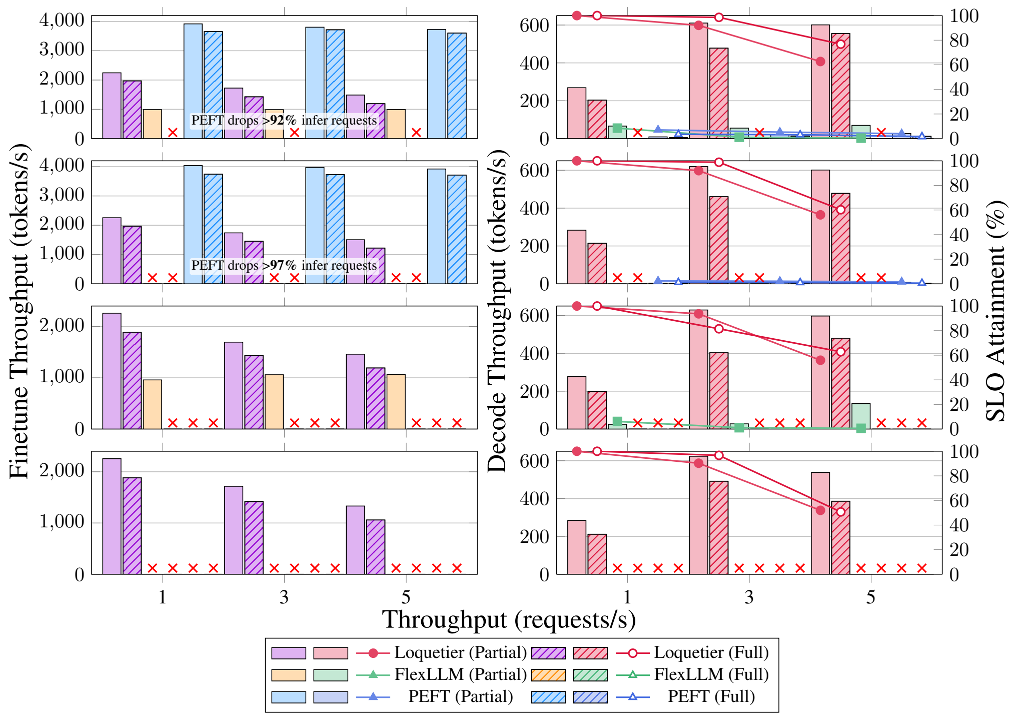

Figure 4: Comparison of the performance of Loquetier and PEFT in unified tasks. The 4 subplots correspond respectively to single-finetune & single-infer, single-finetune & multi-infer, multi-finetune & single-infer, and multi-finetune & multi-infer. The meanings of Partial and Full are the same as in Figure 2. $\times$ indicates that the results were not obtained: FlexLLM and PEFT can only finetune 1 LoRA at a time due to GPU memory limitations, causing it to fail the multi-LoRA fine-tuning scenarios; FlexLLM only support 3 target modules as mentioned in previous figures.

Transformers' batch strategy of padding different inputs to the same length makes its GPU memory footprint greatly affected by the batch size, making it very easy to trigger the error "CUDA out of memory". The processing speed of PEFT is constrained by the batch size to avoid exceeding the GPU memory. In multiple LoRA inference tasks, PEFT can only apply LoRAs in a serial for different configurations of inputs, making the inference speed further degraded. PEFT's SLO attainment rate is unacceptable even under 1 RPS.

Fine-tuning. The test results for the fine-tuning task are shown in Figure 3. The shorter total training time for Loquetier is due to its faster evaluation. Loquetier's fine-tuning is slightly slower than that of PEFT, mainly because of the independent computational calls from the LoRA linears during backward propagation. FlexLLM encounters an unsupported operation error during its peft backward propagation, indicating its inability to complete the experiments. The results show that Loquetier leads to almost no loss of fine-tuning efficiency.

Unified Fine-tuning and Inference. We test a combination of different configurations of fine-tuning and inference tasks, and the results are shown in Figure 4. Loquetier is able to provide an average of about 40% fine-tuning efficiency over three different request arrival rates while maintaining similar SLOs as in the inference-only tasks. PEFT's inference efficiency is too low, resulting in over 90% of the inference tasks timing out before they even begin. PEFT's fine-tuning tasks only drop about 20% efficiency, but this is due to the fact that PEFT has almost no computational overhead on the inference tasks, allowing the vast majority of the computing resources to still be used by the fine-tuning tasks. FlexLLM fails to complete the fine-tuning task, so it is not available for unified tests.

Mutable Capacity Allocation Simulation. In order to evaluate performance facing dynamic loads in real-world scenarios, we design an inference subtask with dynamic input throughput. As shown in Figure 5, Loquetier is able to adaptively adjust the efficiency of both fine-tuning and inference tasks under dynamic loads in the mutable unified task. The fine-tuning task makes concessions for

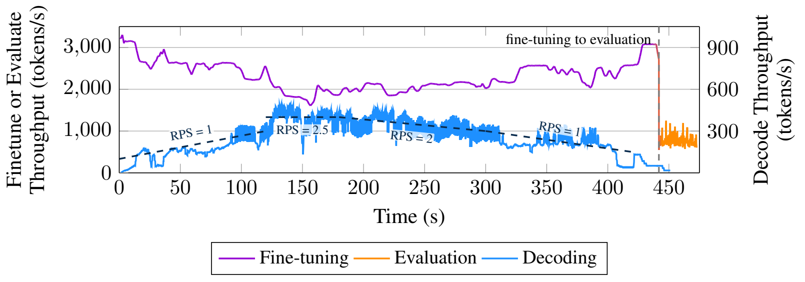

Figure 5: Performance of Loquetier under dynamic load in unified task.

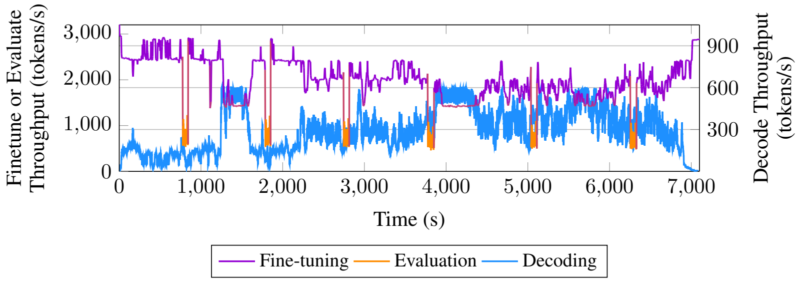

Figure 6: Performance of Loquetier under simulated real-world load in unified task.

the inference task to ensure the quality of service when request throughput increases, and adjusts back the efficiency by itself when throughput decreases.

Simulated Real-world Workload. To better simulate real-world serving conditions, we construct a composite workload using slices from the BurstGPT dataset, as mentioned in Section 4.1. Each slice contains request arrival times, input lengths, and output lengths. In our simulation, we fully utilize the request arrival times and reference the input length data. For an overview of sampling data and preference adjustments, see Appendix D.6. As shown in Figure 6, Loquetier demonstrates strong adaptability to real-world workloads, aligning closely with trends evaluated in the earlier simulation experiments. The final SLO for the entire experiment reaches 92.37%. All requests that failed to meet service metrics occur during transient workload spikes under high-load conditions (RPS > 5), which exceeded the load capacity of the hardware. In all other periods, Loquetier consistently achieves the defined SLO.

5 Conclusion

We present Loquetier, a virtualized multi-LoRA framework that runs fine-tuning and inference tasks uniformly. Loquetier performs well in the inference task, with an SLO attainment 20.8$\times$ higher than PEFT, and up to 3.0$\times$ that of FlexLLM at high request arrival rates, maintaining comparable efficiency in the fine-tuning task. In the unified task, inference efficiency is maintained as much as possible with an SLO attainment 46.4$\times$ higher than PEFT, well balancing the performance of fine-tuning and inference tasks.

Acknowledgments and Disclosure of Funding

We are thankful to the anonymous reviewers for their helpful comments. This work is supported by Jiangsu Science and Technology Major Project (#BG2025005), the National Natural Science Foundation of China (Grants #62172199), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and Jiangsu Wukong Intelligent Computing Digital Technology. Jingwei Xu (`jingweix@nju.edu.cn`) is the corresponding author.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and others. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and others. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and others. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and others. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and others. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, and others. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023a. Publisher: Nature Publishing Group UK London.

Charith Chandra Sai Balne, Sreyoshi Bhaduri, Tamoghna Roy, Vinija Jain, and Aman Chadha. Parameter efficient fine tuning: A comprehensive analysis across applications. *arXiv preprint arXiv:2404.13506*, 2024.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*, 2023a.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 12799–12807, 2023. Issue: 11.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/0cde695b83bd186c1fd456302888454c-Paper-Conference.pdf`.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

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

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*, 2021.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Shih-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard BW Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From lycoris fine-tuning to model evaluation. In *The Twelfth International Conference on Learning Representations*, 2023.

Xiaoyu Kong, Jiancan Wu, An Zhang, Leheng Sheng, Hui Lin, Xiang Wang, and Xiangnan He. Customizing Language Models with Instance-wise LoRA for Sequential Recommendation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 113072–113095. Curran Associates, Inc., 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/cd476d01692c508ddf1cb43c6279a704-Paper-Conference.pdf`.

Jiachen Zhu, Jianghao Lin, Xinyi Dai, Bo Chen, Rong Shan, Jieming Zhu, Ruiming Tang, Yong Yu, and Weinan Zhang. Lifelong personalized low-rank adaptation of large language models for recommendation. *arXiv preprint arXiv:2408.03533*, 2024.

You Zhang, Jin Wang, Liang-Chih Yu, Dan Xu, and Xuejie Zhang. Personalized LoRA for human-centered text understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19588–19596, 2024. Issue: 17.

Yujia Wu, Yiming Shi, Jiwei Wei, Chengwei Sun, Yang Yang, and Heng Tao Shen. Difflora: Generating personalized low-rank adaptation weights with diffusion. *arXiv preprint arXiv:2408.06740*, 2024a.

Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. Llm inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391*, 2024a.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, and others. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*, 2023.

Xupeng Miao, Gabriele Oliaro, Xinhao Cheng, Mengdi Wu, Colin Unger, and Zhihao Jia. FlexLLM: A System for Co-Serving Large Language Model Inference and Parameter-Efficient Finetuning. *arXiv preprint arXiv:2402.18789*, 2024.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 1133–1150, 2025.

Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, and others. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*, 2024.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 640–654, 2024b.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36:65517–65530, 2023.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S^ 3: Increasing GPU Utilization during Generative Inference for Higher Throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and others. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, and others. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. *Proceedings of Machine Learning and Systems*, 6:148–161, 2024.

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

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. *arXiv preprint arXiv:2501.01005*, 2025. URL `https://arxiv.org/abs/2501.01005`.

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, January 2023. URL `https://github.com/NVIDIA/cutlass`.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023a.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*, 2023b.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*, 2023b.

Tangyu Jiang, Haodi Wang, and Chun Yuan. DiffoRA: Enabling Parameter-Efficient LLM Fine-Tuning via Differential Low-Rank Matrix Adaptation. *arXiv preprint arXiv:2502.08905*, 2025.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Yang Li, Shaobo Han, and Shihao Ji. VB-LoRA: extreme parameter efficient fine-tuning with vector banks. *arXiv preprint arXiv:2405.15179*, 2024b.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023b.

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. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A Real-World Workload Dataset to Optimize LLM Serving Systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25)*, Toronto, ON, Canada, 2025. ACM. doi: https://doi.org/10.1145/3711896.3737413. URL `https://doi.org/10.1145/3711896.3737413`.

A Limitations

Loquetier can be improved by combination with other training and fine-tuning optimization methods, and by applying other cluster management systems to provide clusterized services. We plan to provide a backward propagation kernel operating in concert with the SMLM kernel to accelerate fine-tuning.

Co-operating with forward propagation computations during backward propagation can go some way to balancing the bottleneck of GPU memory bandwidth and computational resources for fine-tuning and inference tasks, and Loquetier can be further improved in this regard. We also consider providing support for other LoRA-like PEFT methods.

B Solutions of FlexLLM Backward Procedure Issues

Several cases of operation remain unimplemented in FlexLLM's gradient computation logic, including OP_GELU, OP_RELU, OP_SIGMOID, OP_TANH, and OP_ELU. We noticed that their repository contained forward and backward kernels related to these operations, but they had never applied these backward kernels to their computation flow, resulting in its inability to perform fine-tuning tasks. We ultimately built a runnable version by instantiating these kernels at the missing locations. This fix was based on our understanding of their framework, and we did not implement any additional computational steps. The correction will not result in a performance degradation of FlexLLM.

C Experimental Metrics

We use the following experimental metircs:

- **Service Level Objective (SLO)**: Measuring service satisfaction at a level.
- **SLO Attainment**: Percentage of all requests reaching the given SLO.
- **Request Throughput**: Throughput of incoming inference requests. Measured as request per second (RPS).
- **Decode Throughput**: Throughput of inference requests in decoding. Measured as decode token per second (DTPS).
- **Finetune Throughput**: Throughput of fine-tuning requests in training forward. Measured as finetune token per second (FTPS).
- **Evaluate Throughput**: Throughput of fine-tuning requests in evaluation forward. Measured as evaluate token per second (ETPS).

D Experimental Settings

D.1 SLO

For all inference requests in the experiments, we use the following targets in Figure 3 as SLO.

Since PEFT performs batch generation with padding, resulting in additional computational delays in prefilling and decoding, we do not require more decoding aspects for PEFT.

Table 3: SLO settings.

Framework or System	Max Waiting Time (s)	Mean Decoding Latency (ms)	Max Decoding Latency (ms)
Loquetier	6	200	1,000
PEFT	6	-	-
FlexLLM	6	200	1,000

Table 4: Inference-only tasks configurations. The number of requests in multiple (4) LoRAs is expressed as the total number / number of one LoRA.

	Single (1) LoRA		Multiple (4) LoRAs	
Throughput (RPS)	Requests	Max New Tokens	Requests	Max New Tokens
1	800	400	800 / 200	400
2	1,600	400	1,600 / 400	400
3	2,400	400	2,400 / 600	400
4	3,200	300	3,200 / 800	300
5	4,000	200	4,000 / 500	200

D.2 Inference

We use the following configurations in Figure 4 to test the inference-only tasks.

Note that the maximum tokens supported by FlexLLM is 1024, so all inference requests for FlexLLM do not exceed this limit.

D.3 Fine-tuning

We use the following configurations in Figure 5 to test the fine-tuning-only tasks.

Table 5: Fine-tuning-only tasks configurations.

Configurations	Single (1) LoRA	Multiple (2) LoRAs
LoRA Config		
r (rank)	8	8
lora_alpha	16	16
lora_dropout	0.05	0.05
bias	none	none
task_type	CAUSAL_LM	CAUSAL_LM
init_lora_weights	gaussian	gaussian
Training Args		
per_device_train_batch_size	2	1
per_device_eval_batch_size	2	1
num_train_epochs	4	4
eval_strategy	epoch	epoch
logging_strategy	steps	steps
logging_steps	100	100
save_strategy	epoch	epoch
learning_rate	2e-5	2e-5
gradient_accumulation_steps	4	4
report_to	none	none

D.4 Unified fine-tuning and inference

We use the following configurations in Figure 6 to test the unified tasks.

Note that the maximum tokens supported by FlexLLM is 1024, so all inference requests for FlexLLM do not exceed this limit.

D.5 Mutable capacity allocation simulation

We use the following configurations in Figure 7 for inference requests and the single LoRA fine-tuning configurations in Figure 6 for fine-tuning requests to test the mutable unified tasks.

Table 6: Unified tasks configurations.

	Single (1) LoRA		Multiple (4) LoRAs	
Throughput (RPS)	Requests	Max New Tokens	Requests	Max New Tokens
1	600	400	600 / 150	400
2	1,200	400	1,200 / 300	400
3	1,800	400	1,800 / 450	400
4	2,400	300	2,400 / 600	300
5	3,000	200	3,000 / 750	200

Configurations	Single (1) LoRA	Multiple (2) LoRAs
LoRA Config		
r (rank)	8	8
lora_alpha	16	16
lora_dropout	0.05	0.05
bias	none	none
task_type	CAUSAL_LM	CAUSAL_LM
init_lora_weights	gaussian	gaussian
Training Args		
per_device_train_batch_size	2	1
per_device_eval_batch_size	2	1
num_train_epochs	1	1
eval_strategy	epoch	epoch
logging_strategy	steps	steps
logging_steps	100	100
save_strategy	epoch	epoch
learning_rate	2e-5	2e-5
gradient_accumulation_steps	4	4
report_to	none	none

Table 7: Mutable unified tasks configurations.

	Multiple (4) LoRAs				
Index	LoRA Index	Requests	Throughput (RPS)	Start at (s)	Duration (s)
1	0	120	1	0	120
2	1	150	2.5	120	60
3	2	240	2	180	120
4	3	120	1	300	120

D.6 Simulated real-world workload

We use the following time periods in Figure 8 to test (the inference task of) the simulated real-world workload. The fine-tuning task use the same configuration as mutable capacity allocation simulation.

Based on our analysis of the BurstGPT dataset, less than one-third of the time periods correspond to high-load scenarios, with the majority being low-load periods. Given the lower challenge of low-load periods, we adopt the configuration above to ensure representative coverage. Among the selected workloads, high-load periods include several minutes where the RPS exceeded 5, with a peak RPS of 11.

E Detailed Information Related to S-LoRA in the Experiment

S-LoRA does not support the LLaMA 3 series models, and its repository has been archived. This limitation arises from the Group Query Attention (GQA) architecture used in LLaMA 3, where the shapes of K and V differ from those of Q and O. Consequently, the shape of the weight matrix B in the LoRA linear layers for K and V also differs from Q and O. Current S-LoRA requires all LoRA

Table 8: Time periods configurations. Peak RPS refers to the highest RPS within a 2-second interval.

Time Period	Requests	Mean RPS	Peak RPS
Day 29, 13:00 ~13:20	676	0.563	1.5
Day 29, 15:00 ~15:20	2,145	1.788	11.5
Day 29, 16:00 ~16:20	1,465	1.226	7
Day 33, 13:40 ~14:00	2,823	2.354	10
Day 33, 11:40 ~12:00	2,360	1.966	12
Day 33, 11:00 ~11:20	1,856	1.547	10.5

weights within the same layer to be concatenated at runtime. However, due to the shape discrepancies mentioned above, this concatenation operation fails. As a workaround, we replicate K and V weights in advance during model initialization to enable S-LoRA to start properly.

"Partial" in Figure 2 means that only 4 modules are enabled for S-LoRA including q, k, v, and o, as S-LoRA supports applying LoRA only on these 4 linear layers, and does not support the up, gate, and down layers within the MLP. Therefore, its runtime efficiency resembles the Partial scenario described in our paper, where only three linear layers (up, gate, down) are targeted.

In experiments, we observed instability in the S-LoRA kernel, which frequently produced incorrect outputs leading to NaN or Inf values. These errors propagate quickly, causing model generation failures. At this time, we did not modify the kernel. Our preliminary analysis suggests this may be due to missing synchronization mechanisms in some computational steps. (Note that this is an initial observation and may not be definitive.)

Due to these issues, S-LoRA struggled to complete all inference requests in our scenarios, as it frequently outputs the eos token directly, making SLO appear better than it actually should be.

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.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

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 paper proposes a redesigned kernel SMLM and a unified computation flow for the unified operation of fine-tuning and inference tasks for LoRA models (Section 3.3), and Virtualized Module implementation to isolate PEFT-based model modifications and flexible instance-to-instance migration (Section 3.2).

   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: The limitations of the work is discussed in Section A.

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: This paper does not include theoretical results.

   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: The paper discloses all the information needed for reproducibility (Section 4.1). This paper releases the code for quick reproducibility. The implementation code is available at `https://github.com/s3co3wjy5tr2bdfj/Loquetier`.

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.
- 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: The implementation code is available at `https://github.com/s3co3wjy5tr2bdfj/Loquetier`.

   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, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental settings are presented in Section 4.1. More details could be found in our code repository, which is available at `https://github.com/s3co3wjy5tr2bdfj/Loquetier`.

   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: The experimental metrics are already the statistical result of a large amount of data, so there is no need to conduct multiple experiments to plot the errorbar. For the data in Figure 4, we took a total of 5 data before and after each data point for smoothing operations to improve chart readability and eliminate extreme values over short periods of time.

   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: We present the information on the computer resources in Section 4.1. That is, we test the inference-only tasks on a server with 4 NVIDIA A6000 48G GPUs. We test the fine-tuning-only tasks and the unified fine-tuning and inference tasks on servers with 4 NVIDIA H800 80G GPUs. Each test process has at least 128G of host memory available.

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.
- 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 conformed the NeurIPS Code of Ethics and make sure to preserve anonymity.

   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: [NA]

    Justification: There is no societal impact of the work performed. Our work proposes a framework for unified LLM fine-tuning and serving on multiple LoRA models, which does not lead to any negative societal impacts.

    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: This paper presents a framework for unified LLM fine-tuning and serving with an SMLM kernel for unified tasks, an unified computation flow management, and an implementation of Virtualized Module. This framework does not include any data or models, and therefore this work does not pose any associated risks.

    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: All existing assets used in the paper are listed here. Alpaca: Creative Commons Attribution Non Commercial 4.0, `https://huggingface.co/datasets/tatsu-lab/alpaca`. GSM8K: MIT License, `https://huggingface.co/datasets/openai/gsm8k`. Llama3-8B: Llama 3 Community License Agreement, `https://huggingface.co/meta-llama/Meta-Llama-3-8B`. ShareGPT Vicuna: Apache License Version 2.0, `https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered`.

    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: [Yes]

Justification: Our released code is well organized in the repository.

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]

Justification: This 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: This 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 entire approach 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.