# FELPS: Fair and Efficient Scheduling for Multi-LoRA Serving System

**Yukai Ding**[1]* **Jiawei Jiang**[1,5]* **Chuang Hu**[1] **Xinyan Li**[1] **Susie Xi Rao**[3]
**Shanshan Feng**[1] **Hao Wang**[1] **Fangcheng Fu**[2]† **Xiao Yan**[4]†

[1]School of Computer Science, Wuhan University
[2]School of Artificial Intelligence, Shanghai Jiao Tong University
[3]Bern University of Applied Sciences
[4]Institute for Math and AI, Wuhan University
[5]School of Computer Science, Central China Normal University

{yukai.ding,jiawei.jiang,handc,xinyan_li}@whu.edu.cn
{victor_fengss,wanghao.cs,yanxiaosunny}@whu.edu.cn
ccchengff@sjtu.edu.cn susie.rao@bfh.ch

## Abstract

Low-Rank Adaptation (LoRA) is a popular parameter-efficient fine-tuning method for Large Language Models (LLMs), and practical systems usually need to serve many concurrent LoRA adapters to process different tasks. This is challenging due to the inherent tension between *fairness* and *efficiency*, i.e., achieving fairness requires frequent switching among the adapters such that they have similar service quality, while good efficiency requires minimizing adapter switching to reduce memory movement overheads. Existing schedulers can only achieve a fixed trade-off between fairness and efficiency, and their trade-offs are often suboptimal. To tackle this problem, we propose **F**airness-aware **E**fficient **L**oRA **P**riority **S**cheduling (FELPS), which explicitly decomposes fairness and efficiency considerations. In particular, for fairness, FELPS adapts the classical principle of proportional fairness and ensures that the amount of service received by each adapter is proportional to its workload. For efficiency, FELPS prioritizes adapters that are already loaded into GPU memory. To conduct scheduling, FELPS combines the fairness and efficiency terms with an adjustable weight to achieve arbitrary trade-offs between fairness and efficiency. Evaluations on real workloads show FELPS's superior trade-off between fairness and efficiency over all baselines.

## 1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities on general artificial intelligence tasks [Achiam et al., 2023, OpenAI, 2024, Yang et al., 2024]. For domain-specific applications (e.g., finance and medical), fine-tuning is usually employed to adapt pre-trained LLMs to the target tasks [Liu et al., 2023, Singhal et al., 2023, Xu et al., 2024]. To enhance efficiency, various parameter-efficient fine-tuning (PEFT) methods have been proposed [Houlsby et al., 2019, Li and Liang, 2021, Zhou et al., 2022]. Among them, Low-Rank Adaptation (LoRA) [Hu et al., 2022] is one of the most popular and representative approaches, which introduces trainable low-rank matrices to adapt the parameters of an LLM. In particular, each layer of a LoRA adapter can be expressed as

$$W_a = W_{d\times d'} + A_{d\times r}B_{r\times d'},$$

*Equal contribution.
†Corresponding authors: Fangcheng Fu (ccchengff@sjtu.edu.cn) and Xiao Yan (yanxiaosunny@whu.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: Capability comparison of existing schedulers ($\surd$ = supported, $\triangle$ = partial, $\times$ = not addressed).

| | FCFS | VTC | AC | Inlaid | FELPS |
|---|---|---|---|---|---|
| Fairness (Starvation Awareness) | $\triangle$ | $\surd$ | $\times$ | $\triangle$ | $\surd$ |
| Efficiency (Memory Locality) | $\times$ | $\times$ | $\surd$ | $\surd$ | $\surd$ |
| Tunable trade-off | $\times$ | $\times$ | $\triangle$ | $\triangle$ | $\surd$ |

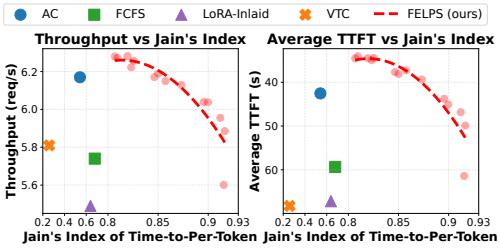



Figure 1: Comparing FELPS with representative baselines. Jain's Index measures per-adapter service fairness via normalized time-to-per-token. Higher is better on both axes.

where $W$ is the weight matrix of the LLM, and $A$ and $B$ are the adapter weights with $r \ll d, d'$.

**Existing Solutions and Their Limitations.** Practical systems usually serve many concurrent LoRA adapters to handle different tasks [Sheng et al., 2023, Chen et al., 2024]. That is, since each request requires a specific adapter, concurrent serving necessitates frequent adapter loading and switching. Serving systems usually target *fairness* and *efficiency* but the two goals are conflicting in this scenario. In particular, fairness means that requests for all adapters should have similar quality of service (QoS), as measured by latency metrics such as Time-to-Per-Token, and this requires the system to frequently switch the adapters to avoid starvation. However, this introduces high memory movement overheads (i.e., between CPU and GPU) and exacerbates the memory contention between adapter weights and KV caches, thereby degrading efficiency. Conversely, efficiency exploits memory locality: serving fewer concurrent adapters reduces switching overhead and reserves GPU memory for KV caches, but inevitably starves requests targeting unloaded adapters.

Existing schedulers can only achieve *fixed* and *suboptimal* trade-offs between fairness and efficiency as shown in Table 1. Specifically, FCFS [Silberschatz et al., 2019] offers implicit fairness via arrival ordering but ignores memory locality, causing frequent adapter switching. Designed for multi-tenant scenarios, VTC [Sheng et al., 2024] enforces strict token-level fairness via round-robin, yet sacrifices efficiency by frequent context switching and memory fragmentation. Adapter Clustering (AC) [Sheng et al., 2023] and LoRA-Inlaid [Xia et al., 2024] optimize efficiency through memory locality by limiting active adapters during inference, with Inlaid additionally leveraging length prediction for SRTF. However, neither provides fairness guarantees, and Inlaid only partially mitigates starvation via a timeout threshold. The limitations of existing schedulers lead us to the following research question:

> *Can we design a scheduler to explore different trade-offs between fairness and efficiency and improve the trade-offs made by existing schedulers?*

**Our solution.** To resolve this conflict, we propose **F**airness-aware **E**fficient **L**oRA **P**riority **S**cheduling (FELPS), a dynamic scheduling strategy designed to navigate the trade-off between efficiency and fairness. Unlike the strict arrival-based fairness of FCFS or the absolute resource fairness of Round-Robin, FELPS is fundamentally built upon the principle of **Proportional-Fair Scheduling (PFS)** [Kelly, 1997, Tse, 1999]: each adapter should receive service proportional to its demand.

Specifically, for fairness, FELPS calculates a real-time waiting cost for each LoRA task by weighting the accumulated waiting time and request lengths, and it maintains the historical service usage using an Exponential Moving Average (EMA). The *Service-aware Fair Priority* is then derived by contrasting this waiting cost against the historical service usage. For efficiency, FELPS introduces a *Locality-aware Penalty* for currently unloaded LoRA tasks to encourage reuse of active adapters, with an adjustable weight to navigate the fairness-efficiency trade-off. Our theoretical analysis demonstrates that this design guarantees equitable resource distribution among tasks in the long run.

We conduct comprehensive experiments using realistic workloads constructed from public datasets. Figure 1 compares FELPS against representative baselines on the multi-task workload. Baselines like FCFS and AC achieve suboptimal trade-offs, prioritizing only a single dimension of fairness or efficiency. In contrast, FELPS achieves a superior balance, forming the optimal Pareto frontier as illustrated in the figure. Among all experiments, compared to the efficiency-oriented baseline **AC**, FELPS improves fairness (Jain's Index) by over **10 percentage points** while maintaining comparable throughput. Conversely, compared to the fairness-oriented baseline **FCFS**, FELPS boosts throughput

by approximately **10%** and significantly reduces average TTFT by over **20%**. These results confirm that FELPS effectively resolves the efficiency-fairness tension in multi-LoRA serving systems.

To summarize, our contributions are as follows:

- We identify that the primary conflict between fairness and efficiency in multi-LoRA serving stems from adapter switching overheads and memory contention with KV caches. We reveal that existing works are limited to a singular focus, either on fairness or efficiency.
- We propose **FELPS**, a scheduling framework based on proportional fairness. Unlike rigid baselines, FELPS offers a tunable mechanism that enables explicit and arbitrary trade-offs between fairness and efficiency.
- We design a prioritization mechanism that synthesizes *Service-aware Fair Priority* and *Locality-aware Penalty*. Theoretical analysis shows that our design guarantees fairness in long-term resource allocation.

## 2 Related Work

**LLM Serving and Scheduling.** Foundational LLM serving systems like Orca [Yu et al., 2022] and vLLM [Kwon et al., 2023] use iteration-level scheduling and PagedAttention with FCFS policies. FastServe [Wu et al., 2023] and Fu et al. [2024] optimize JCT via MLFQ and SRTF with length prediction. For multi-LoRA serving, S-LoRA [Sheng et al., 2023] and Punica [Chen et al., 2024] share a base model across adapters; S-LoRA limits active adapters (AC) for throughput while Punica uses FCFS. LoRA-Inlaid [Xia et al., 2024] combines SRTF with request grouping. VTC [Sheng et al., 2024] enforces fairness via virtual-token round-robin at the cost of throughput. These works each optimize a single objective, yielding a fixed efficiency-fairness trade-off.

**Cluster-Level Multi-LoRA Serving System.** CaraServe [Li et al., 2024] uses CPU-assisted rank-aware scheduling to meet SLO targets across GPU nodes. dLoRA [Wu et al., 2024] improves load balance via proactive dispatching (dynamic estimation of queueing and I/O time) and cross-node cooperative migration. LoRAServe [Jaiswal et al., 2025] co-optimizes adapter placement and cluster routing to reduce rank heterogeneity per server. These works operate at the *macro* inter-node routing and placement layer; FELPS operates at the *micro* intra-node scheduling layer. The two are orthogonal: requests routed by cluster-level frameworks can be locally dispatched by FELPS to guarantee per-node fairness and efficiency.

## 3 Preliminaries

We consider a multi-task serving system hosting a set of LoRA adapters $\mathcal{A} = \{1, \ldots, N\}$ on a shared Large Language Model (LLM). The system operates via two concurrent asynchronous flows: a *Monitoring Stream* for request ingestion and an *Execution Stream* for continuous serving.

**Monitoring Stream.** Requests arrive randomly from different LoRA tasks. Each request is denoted as a tuple $r = (t_r, x_r, a_r)$, comprising an arrival timestamp $t_r$, an input token sequence $x_r$, and a target adapter identifier $a_r \in \mathcal{A}$. The target adapter is assigned upstream and remains fixed throughout execution. The monitoring stream continuously supervises the incoming traffic. Upon the arrival of a request $r$, the system first employs a lightweight proxy model (e.g., BERT [Devlin et al., 2019] or OPT [Zhang et al., 2022]) to predict its output length $|\hat{y}_r|$. Subsequently, $r$ is appended to the waiting queue $\mathcal{Q}_{a_r}$ associated with its assigned adapter.

**Execution Stream.** In parallel, the execution stream maintains a running batch $\mathcal{B}$ and performs inference following the continuous batching paradigm. At discrete scheduling intervals (e.g., after a fixed number of decoding steps), the system selects candidate requests from $\{\mathcal{Q}_a\}$ based on their calculated priorities and attempts to move them into $\mathcal{B}$ for the prefill phase, loading the corresponding LoRA adapters into GPU memory if necessary. To ensure system stability, the construction of a candidate batch must strictly satisfy the following resource constraints:

- **Memory Constraint:** The total memory consumption, including both Key-Value (KV) cache and loaded LoRA adapter weights, must not exceed the pre-allocated pool size $M_{\text{pool}}$:

$$\mathcal{M}_{\text{kv}} + \mathcal{M}_{\text{lora}} \leq M_{\text{pool}} \tag{1}$$

- **Context Length Constraint:** Every request in the batch must fit within the model's maximum context window $L_{\max}$, considering both input and estimated output lengths:

$$|x_r| + |y_r| \leq L_{\max}, \quad \forall r \in \mathcal{B} \tag{2}$$

After each iteration, the system streams the generated responses back to the users. Requests that have completed generation are immediately evicted from $\mathcal{B}$ to release memory resources.

# 4 FELPS Scheduler

In this section, we present our proposed method, **FELPS** (**F**airness-aware **E**fficient **L**oRA **P**riority **S**cheduling). We first formalize the fairness objective, then elaborate on our core priority design that mathematically synthesizes fairness and efficiency, and finally detail the system workflow.

## 4.1 Service and Locality-Aware Priority Design

### 4.1.1 Service-aware Fair Priority.

We formalize the fairness objective as *demand-proportional service allocation*. Let $\omega_a = \lambda_a \cdot \bar{L}_a$ denote the workload arrival rate (tokens/sec) of adapter $a$, where $\lambda_a$ is the request rate and $\bar{L}_a$ is the mean token length, and let $\bar{\mu}_a$ be its long-run average service rate. The system is fair if each adapter receives service in proportion to its demand:

$$\frac{\bar{\mu}_i}{\bar{\mu}_j} = \frac{\omega_i}{\omega_j}, \quad \forall i, j \in \mathcal{A}. \tag{3}$$

To realize this, we draw on Proportional-Fair Scheduling (PFS) [Kelly, 1997], which maximizes the aggregate logarithmic utility $\sum_{a \in \mathcal{A}} \log(\bar{R}_a)$ by scheduling the user maximizing $R_a(t)/\bar{R}_a(t)$ at each decision instant. Adapting this to multi-LoRA serving, we propose the *Service-aware Fair Priority (SFP)*: we reinterpret the instantaneous rate as the accumulated waiting cost $W_a(t)$ and the historical throughput as the smoothed GPU service history $\tilde{S}_a(t)$, prioritizing adapters overloaded relative to their recent resource consumption, thereby mitigating starvation and ensuring fairness.

**Waiting Cost.** Let $\mathcal{Q}_a(t) = \{r \mid a_r = a\}$ be the set of pending requests for adapter $a$. Following VTC [Sheng et al., 2024], the total waiting cost $W_a(t)$ weights the elapsed time of each request by its estimated computational load:

$$W_a(t) = \sum_{r \in \mathcal{Q}_a(t)} (t - t_r) \cdot L_r, L_r = |x_r| + \rho \cdot |\hat{y}_r|, \tag{4}$$

where $t_r$ is the arrival time, $|x_r|$ and $|\hat{y}_r|$ are the input and estimated output lengths, and $\rho$ is the per-token latency ratio between the decoding and prefill phases.

**Service History.** For the denominator, we adopt a time-dependent Exponential Moving Average (EMA) to track the service history, preventing outdated records from dominating decisions. Let $\tilde{S}_a(t)$ denote the smoothed accumulated service time for LoRA $a$. The update rule is formulated as:

$$\tilde{S}_a(t) = \alpha_t \cdot \tilde{S}_a(t - \Delta t) + (1 - \alpha_t) \cdot \Delta S_a(t), \tag{5}$$

where $\Delta S_a(t)$ is the actual GPU time consumed by LoRA $a$ during the interval $\Delta t$ since the last update. The decay factor $\alpha_t$ adjusts dynamically based on the elapsed time to maintain a consistent time constant $T_{\text{ema}}$:

$$\alpha_t = \exp\left(-\frac{\Delta t}{T_{\text{ema}}}\right). \tag{6}$$

To prevent accumulated service history from unfairly penalizing tail requests after long idle periods, $\tilde{S}_a(t)$ is reset to 0 if the time since the last update exceeds a set threshold.

Finally, the Service-Aware Fair Priority for adapter $a$ is defined as:

$$SFP_a(t) = \frac{W_a(t)}{\tilde{S}_a(t) + \epsilon}, \tag{7}$$

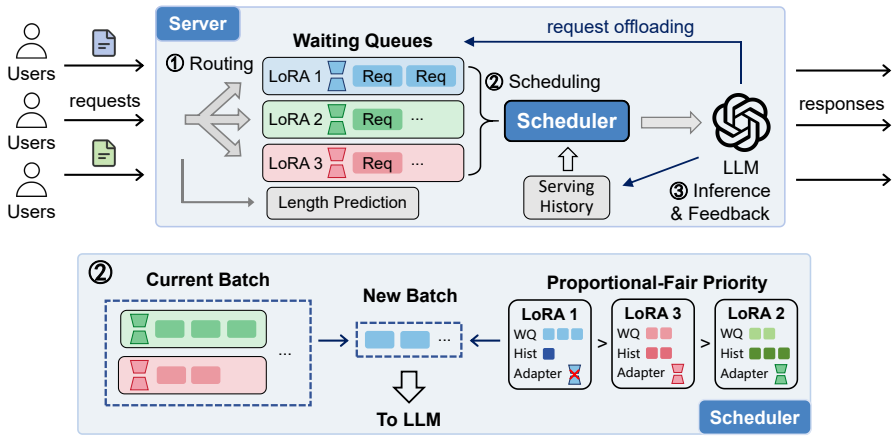



Figure 2: Overview of the FELPS scheduling workflow.

where $\epsilon$ is a small constant to ensure numerical stability during the system's initialization phase.

**Theoretical Analysis.** We analyze the long-run fairness of SFP in Appendix A. Under the fluid-limit model with strict overload ($\Lambda = \sum_a \omega_a > \mu_{\max}$) and work-conserving max-SFP scheduling, the long-run service rates $\bar{\mu}_a$ converge to a demand-proportional allocation:

$$\lim_{t\to\infty} \frac{\bar{\mu}_i}{\bar{\mu}_j} = \frac{\omega_i}{\omega_j}, \quad \forall i, j \in \mathcal{B}, \tag{8}$$

where $\omega_a = \lambda_a \bar{L}_a$ is the workload arrival rate of adapter $a$ and $\mathcal{B} = \{a : \bar{\mu}_a < \omega_a\}$ is the bottleneck adapter set. Adapters outside $\mathcal{B}$ ($\bar{\mu}_a \geq \omega_a$) correspond to sporadic low-frequency tasks whose queues are rapidly drained under SFP and do not contribute to sustained overload analysis.

#### 4.1.2 Locality-aware Penalty.

**Overhead of Adapter Switching.** While the SFP effectively captures fairness by favoring tasks with more pending requests and lower recent service, it overlooks the system-level memory constraints. In practice, maximizing throughput requires the scheduler to consider resource utilization, particularly GPU memory, which imposes significant constraints in high-concurrency environments. Specifically, maintaining a high number of active adapters limits the memory available for KV cache, directly reducing the effective batch size and consequently impairing system throughput.

**Solution: A Penalty Mechanism.** To address this issue, we introduce a *Locality-aware Penalty (LP)* mechanism. Instead of an additive priority adjustment, we apply a multiplicative penalty factor to the priority of adapters that are not currently loaded in the GPU memory. The goal is to discourage the scheduling of unloaded adapters when resource contention is high, unless their accumulated waiting cost is sufficiently high to justify the swapping cost.

We define a penalty coefficient $\beta_a$ for each LoRA adapter $a$. If an adapter is already loaded in the memory, no penalty is applied (i.e., $\beta_a = 1$). Conversely, for unloaded adapters, $\beta_a$ scales with the adapter's size. Since an adapter's GPU memory footprint is proportional to its parameter count, we approximate this cost using the adapter's rank. To ensure the penalty remains within a reasonable range and does not strictly prohibit the service of high-rank tasks, we adopt a logarithmic scaling formulation. Formally, the locality-aware penalty coefficient $\beta_a$ is defined as:

$$\beta_a = \begin{cases} 1, & \text{if } a \text{ is already loaded;} \\ 1 + \lambda \cdot \log_2\left(1 + \frac{rank_a}{32}\right), & \text{if } a \text{ is not loaded.} \end{cases} \tag{9}$$

Here, $\lambda$ is a hyperparameter that controls the degree of penalty, and the constant 32 serves as a normalization factor for the rank, ensuring that standard LoRA configurations (e.g., $rank = 32$ or 64) incur a moderate penalty, while larger LoRA adapters incur a progressively higher penalty.

Integrating this penalty with the fairness metric, the final scheduling priority $P_a(t)$ is calculated by adjusting $\beta_a$ to the Service-Aware Fair Priority ($SFP_a$):

$$P_a(t) = \frac{SFP_a(t)}{\beta_a}. \tag{10}$$

This division-based formulation ensures that requests requiring a new adapter load must have accumulated a proportionally higher waiting cost to overcome the locality penalty. This mechanism effectively encourages the system to aggregate requests for currently loaded adapters within the same batch, thereby maximizing token generation throughput under resource constraints without permanently starving unloaded tasks and avoiding frequent adapter switching.

**Theoretical Analysis.** We extend the analysis to the penalty-weighted case in Appendix B. Assuming a micro-scheduling limit cycle where each adapter alternates between loaded ($\beta_a = 1$) and waiting ($\beta_a = \beta_{\text{high},a}$) phases with duty cycle $\mathcal{D}_a$, the time-averaged inverse penalty is $\beta_{\text{eff},a} = \beta_{\text{high},a}/(1 + \mathcal{D}_a(\beta_{\text{high},a} - 1))$. The allocation ratio is then multiplicatively bounded:

$$\frac{\bar{\rho}_i}{\bar{\rho}_j} = \frac{\omega_i}{\omega_j} \cdot \frac{\beta_{\text{eff},j}}{\beta_{\text{eff},i}} \cdot \gamma_{ij}, \qquad \gamma_{ij} \in \left(B^{-1},\ B\right), \tag{11}$$

where $B = \max_{a,b \in \mathcal{B}} \beta_{\text{eff},a}/\beta_{\text{eff},b}$ is bounded by the configured penalty range (e.g., $B \leq 2.6$ for typical $\lambda = 1$, rank $\leq 64$).

### 4.2 Implementation and Scheduling Workflow

**Length Prediction.** Following the practice of recent works, we employ a lightweight proxy model `OPT-125M` to estimate the generation length. The length prediction is formulated as a **classification task**. We discretize the output length into coarse-grained buckets with a bucket width of 100 tokens. The proxy model takes the input prompt $x_r$ and predicts the probability distribution over these buckets. The upper bound of the predicted bucket is then used as the estimated output length $|\hat{y}_r|$.

**Memory Protection and Admission Control.** FELPS builds on S-LoRA [Sheng et al., 2023], inheriting its unified paging memory management and tensor parallelism. We implement **Fine-grained Admission Control (FAC)**, which reserves memory for $\min(L_{\text{remain}}, \Delta_{\text{step}})$ rather than the entire predicted output length, admitting requests only when the estimated peak memory usage stays within $M_{\text{pool}}$. Reactive offloading handles memory shortages, including those caused by underestimated output lengths, by preempting and offloading the request with the shortest total length. This safeguard remains enabled both with and without FAC.

**Scheduling Workflow.** Figure 2 illustrates the scheduling workflow of FELPS, which operates in three phases:

**1) Routing Phase.** Upon the arrival of a request, the system employs a lightweight length predictor to estimate its output token length for cost estimation. The request is then appended to the waiting queue corresponding to its pre-assigned target LoRA adapter.

**2) Scheduling Phase.** The scheduler periodically constructs a new execution batch from the pending requests. As depicted in the bottom panel, this process is driven by the *Proportional-Fair Priority* mechanism. The scheduler evaluates candidate LoRA tasks by combining their waiting cost, service history, and adapter locality and prioritizing requests with the highest priority to form the new batch.

**3) Inference and Feedback Phase.** The selected batch is submitted to the shared LLM for inference. During execution, the system maintains a closed feedback loop: generated responses are streamed to users, and the GPU service time is fed back to update the serving history for future priority calibration.

### 4.3 Discussion

**Distinct from classical proportional fairness.** Classical PF [Kelly, 1997] allocates continuously divisible bandwidth via the ratio of instantaneous rate to historical throughput. Multi-LoRA serving, by contrast, involves discrete GPU time slots and indivisible adapter loading, rendering the classical formulation inapplicable. FELPS restructures the priority as *cumulative waiting cost / historical service* (Eq. 7) and decouples a locality-aware penalty (Eq. 9) to handle GPU memory contention

between adapter weights and KV caches, a constraint unique to the multi-LoRA setting. Together they achieve a superior trade-off between fairness and efficiency.

**Scalability.** FELPS operates at the intra-node temporal scheduling layer and is orthogonal to cluster-level orchestration frameworks such as CaraServe and LoRAServe. In a multi-GPU deployment, a cluster router can globally dispatch requests to nodes, each running an independent FELPS instance to guarantee per-node fairness and efficiency. Extending FELPS to jointly optimize across nodes is an important direction for future work.

**Limitations.** FELPS relies on a lightweight length predictor: while our robustness experiments (Appendix D.2) show graceful degradation under noise, extreme prediction errors could distort priority ranking. Meanwhile, the penalty weight $\lambda$ provides a controllable monotonic fairness-efficiency knob, but its optimal value depends on workload characteristics and currently requires offline calibration.

# 5 Experiment

In this section, we evaluate our proposed method FELPS under various workloads compared to existing baselines. The results demonstrate that our approach outperforms baselines in terms of system throughput while simultaneously maintaining an optimal trade-off in fairness.

## 5.1 Experiment Setup

**Testbeds and Models.** To evaluate system performance across different scales, we utilize two distinct hardware-model configurations: (1) **Llama-2-7B** [Touvron et al., 2023] deployed on a single NVIDIA RTX 4090 GPU (24GB memory), and (2) **Llama-2-13B** [Touvron et al., 2023] deployed on a single NVIDIA RTX A6000 GPU (48GB memory). For both setups, we employ LoRA adapters with varying ranks (32 and 64) to simulate diverse memory footprints and computational costs, thereby testing the scheduler's robustness under heterogeneous model configurations.

**Datasets and Workloads.** Our evaluation employs two primary workloads: *Multi-task* and *Conversation*. For the *Multi-task* workload, we aggregate public datasets [Pham, 2023, Xu et al., 2025, Fan et al., 2018, Peng et al., 2023, See et al., 2017, Zhao et al., 2023, Cobbe et al., 2021, Zhang et al., 2020, Hendrycks et al., 2021] spanning tasks such as code generation, reasoning, and summarization, mapping them to distinct LoRA adapters. To simulate realistic production traffic, we synthesize request arrival times via a Gamma process and task popularity using a Zipf distribution.

For the *Conversation* workload, we utilize the Chatbot Arena Conversations dataset [Zheng et al., 2023], mapping its 20 source LLMs to corresponding LoRA adapters. Leveraging the dataset's inherent real-world timestamps for arrival patterns, we simply sample tasks randomly to maintain the realistic distribution. Details are provided in Appendix C.

**Baselines.** We evaluate FELPS against four representative baselines: the standard **FCFS** [Silberschatz et al., 2019], which schedules requests strictly by arrival order; the throughput-oriented **AC** [Sheng et al., 2023], which limits concurrent adapter loading to enforce reuse and minimize switching; the latency-optimized **Inlaid** (LoRA-Inlaid) [Xia et al., 2024], which combines SRTF logic with a concurrent adapter limit and starvation prevention mechanisms; and the fairness-centric **VTC** [Sheng et al., 2024], which prioritizes tasks with the lowest accumulated token usage.

**Evaluation Metrics.** To comprehensively evaluate system performance, we employ **Throughput** (req/s) to measure overall processing capacity, alongside **Average** and **P95 Time-To-First-Token (TTFT)** (s) to assess response latency and tail delays. Regarding fairness, we use **Jain's Fairness Index** [Jain et al., 1984] on each adapter's average **time-to-per-token** (TTPT), $\tau_i$, to evaluate fairness from the user-latency perspective. Here, $\tau_i$ is similarly normalized by the weighted token length following Eq. 4. Jain's Index is defined as $\mathcal{J} = \frac{(\sum_{i=1}^{N} \tau_i)^2}{N \cdot \sum_{i=1}^{N} \tau_i^2}$ for $N$ adapters, where $\mathcal{J} \to 1$ indicates superior fairness. Appendix D.4 additionally reports Jain's index of per-adapter service-to-demand ratios over time, evaluating demand-proportional service allocation in connection with our theoretical analysis.

Table 2: Overall performance of all methods on *Conversation* and *Multi-task* workloads. The top-2 results are highlighted for each workload. Tput: Throughput (req/s), Avg TTFT: Average TTFT (s), P95 TTFT (s), Jain: Jain's Fairness Index (%).

| Model | Method | Multi-task | | | | Conversation | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Tput | Avg TTFT | P95 TTFT | Jain (%) | Tput | Avg TTFT | P95 TTFT | Jain (%) |
| Llama-2-7B | FCFS | 5.74 | 59.35 | 102.27 | 67.85 | 5.21 | 85.85 | 151.40 | 86.34 |
| | VTC | 5.81 | 68.26 | 199.91 | 25.87 | 5.24 | 82.49 | 198.04 | 85.78 |
| | AC | 6.21 | 42.32 | 103.29 | 48.59 | 5.65 | 69.97 | 149.88 | 83.54 |
| | Inlaid | 5.49 | 67.19 | 121.18 | 63.92 | 4.95 | 97.17 | 174.28 | 82.81 |
| | FELPS | **6.28** | **34.48** | **73.05** | **81.95** | **5.73** | **59.81** | **137.93** | **97.03** |
| Llama-2-13B | FCFS | 3.08 | 83.16 | 154.30 | 84.50 | 3.14 | 80.34 | **139.42** | 78.33 |
| | VTC | 3.10 | 105.31 | 287.63 | 46.76 | 3.08 | 82.77 | 207.65 | **90.09** |
| | AC | **3.29** | **64.93** | 184.63 | 74.48 | **3.35** | 73.02 | 150.87 | 77.54 |
| | Inlaid | 2.95 | 99.15 | 174.73 | 83.72 | 2.88 | 103.83 | 180.91 | 77.33 |
| | FELPS | 3.25 | 65.40 | **136.27** | **87.87** | 3.27 | **64.46** | 182.63 | 88.56 |

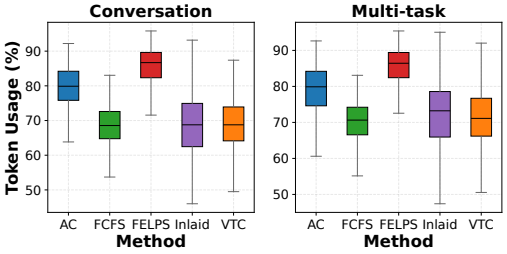



Figure 3: Token usage across iterations during serving. Higher token usage means better efficiency.

Table 3: Ablation Study on FELPS.

| Method | Tput | Avg TTFT | P95 TTFT | Jain (%) |
|---|---|---|---|---|
| *Multi-task* | | | | |
| FELPS | 6.28 | 34.48 | 73.05 | 81.95 |
| w/o EMA | 6.38 | 33.62 | 83.94 | 82.11 |
| w/o SFP | 6.19 | 40.59 | 232.81 | 29.09 |
| w/o LP | 5.60 | 61.42 | 113.54 | 91.55 |
| w/o FAC | 6.34 | 33.98 | 87.46 | 66.68 |
| *Conversation* | | | | |
| FELPS | 5.73 | 59.81 | 137.93 | 97.03 |
| w/o EMA | 5.83 | 66.72 | 204.89 | 83.37 |
| w/o SFP | 5.90 | 59.12 | 241.51 | 55.26 |
| w/o LP | 5.02 | 92.13 | 203.69 | 97.00 |
| w/o FAC | 5.87 | 64.91 | 170.35 | 93.31 |

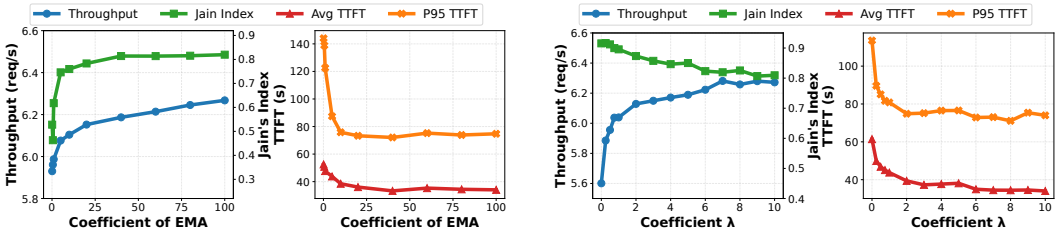
### 5.2 Main Experiment

We evaluate the performance of FELPS against all baselines under *Multi-task* and *Conversation* workloads. Table 2 presents the overall performance comparison of FELPS against all baselines across models and workloads. The results demonstrate that FELPS successfully breaks the inherent trade-off between system efficiency and service fairness, achieving the best comprehensive performance.

**Efficiency-Fairness Trade-off.** Existing methods often sacrifice one metric to optimize another. For instance, **AC**, which prioritizes maximizing throughput via aggressive aggregation, achieves high throughput (e.g., 6.21 req/s in Llama-2-7B multi-task) but suffers from poor fairness (Jain's Index of 48.59%). Conversely, **FCFS** maintains reasonable fairness (67.85%) but lags in throughput and latency. Similarly, **VTC** enforces absolute fairness, which inadvertently delays high-weight tasks, resulting in the highest P95 TTFT and the worst Jain's Index in multi-task. **Inlaid** is constrained by its SRTF policy, which induces starvation for long tasks and relies heavily on the accuracy of length prediction, leading to poor performance in Average TTFT.

In contrast, **FELPS** achieves a significantly better trade-off: on Llama-2-7B multi-task, it not only surpasses AC in throughput but also improves Jain's Index by 33.36 percentage points (pp). This indicates that FELPS's priority mechanism effectively identifies urgent tasks without compromising system capacity. This efficiency stems from optimal memory utilization as shown in Figure 3. FELPS consistently maintains the highest token usage rate, fully exploiting the GPU memory pool rather than remaining idle.

**Latency and Stability.** FELPS consistently delivers the lowest Average TTFT and competitive P95 TTFT across most settings, achieving a $\sim$20% reduction over AC on Llama-2-7B multi-task. Regarding tail latency (P95 TTFT), on Llama-2-13B multi-task, FELPS achieves 136.27s, outperforming FCFS (154.30s) and AC (184.63s). This proves that the locality-aware penalty and EMA-based history tracking effectively prevent starvation and thrashing. On Llama-2-13B conversation, FELPS exhibits higher P95 TTFT than FCFS. Relatively uniform task popularity and the larger GPU memory




(a) Impact of EMA coefficient $T_{ema}$.

(b) Impact of locality-penalty coefficient $\lambda$.

Figure 4: Sensitivity analysis of FELPS under the multi-task workload on Llama-2-7B.

capacity reduce memory contention, favoring FCFS-style arrival-order scheduling. In this setting, FELPS's cross-adapter service balancing can shift its behavior toward VTC-style round-robin scheduling, sacrificing tail latency. Nevertheless, FELPS achieves the lowest average TTFT while maintaining competitive throughput and fairness.

**Effect Across Model Size.** FELPS retains its advantages as the model size scales. On Llama-2-13B conversation, while AC achieves marginally higher throughput, it comes at a significant cost to fairness (77.54% vs. 88.56%). FELPS maintains a high fairness score ($> 87\%$) and low average TTFT across all 13B scenarios, confirming its robustness in resource-constrained environments.

### 5.3 Ablation Study

We further conduct ablation studies on Llama-2-7B with *Multi-task* and *Conversation* workloads. As shown in Table 3, we investigate the impact of the EMA component, the Service-aware Fair Priority (SFP), the Locality-aware Penalty (LP), and the Fine-grained Admission Control (FAC).

**Impact of EMA.** Replacing EMA with long-term accumulation yields marginal throughput gains but severely compromises stability, especially in the conversation workload. P95 TTFT deteriorates (137s $\rightarrow$ 204s), and Jain's Index drops by over 12 pp. This confirms that EMA is crucial for *cycle fairness*: by discounting outdated history, it prevents excessive adjustments for past starvation.

**Impact of SFP and LP.** Removing SFP causes a fairness collapse across all workloads: Jain's Index plummets to 29.09% in multi-task and 55.26% in conversation, with P95 TTFT tripling due to severe starvation. Conversely, disabling LP triggers a consistent throughput drop ($\sim$11%) as the scheduler incurs excessive adapter switching. This trade-off confirms the complementary roles of the two components: SFP enforces proportional fairness as the foundational priority, while LP modulates the resulting allocation to preserve system efficiency by curbing unnecessary switching.

**Impact of FAC.** The w/o FAC variant reserves memory for the entire predicted output length while retaining SFP, LP, and reactive offloading. On multi-task, it slightly increases throughput (6.28 to 6.34 req/s) but reduces Jain's index from 81.95% to 66.68% and worsens P95 TTFT. Its more conservative admission may reduce offloading overhead, favoring throughput. In contrast, FAC allows more requests prioritized by SFP to enter the batch, supporting timely service for underserved adapters, potentially at the cost of increased offloading.

### 5.4 Sensitivity Experiment

We further conduct sensitivity experiments on FELPS, focusing on the EMA Coefficient $T_{ema}$ and the Locality-penalty Coefficient $\lambda$, using the multi-task workload on Llama-2-7B.

**Impact of EMA Coefficient $T_{ema}$.** Figure 4a illustrates the sensitivity of FELPS to the EMA coefficient $T_{ema}$. With small $T_{ema}$ ($< 20$), the scheduler over-reacts to short-term fluctuations, causing unstable priority oscillations, peaked P95 TTFT ($\sim$ 140s), and poor fairness (Jain's Index $\sim 0.5$). Conversely, increasing $T_{ema}$ smooths transient noise and balances resources over a longer

horizon. At $T_{ema} > 40$, both fairness (Jain's Index $> 0.8$) and throughput ($> 6.2$ req/s) stabilize, confirming that a sufficiently large history window is essential for robust long-term scheduling.

**Impact of Locality-penalty Coefficient $\lambda$.** Figure 4b illustrates the sensitivity to the penalty coefficient $\lambda$. At $\lambda \approx 0$, the system prioritizes strict fairness ($> 0.9$) but suffers from severe thrashing and low throughput ($\sim 5.6$ req/s). Increasing $\lambda$ effectively penalizes switching, boosting throughput to $\sim 6.2$ req/s and reducing P95 TTFT to $\sim 75$s, while fairness stabilizes at $\sim 0.8$. This confirms that a moderate $\lambda$ achieves an optimal balance, trading marginal fairness for significant efficiency gains.

**Additional Evaluation.** Appendix D provides further evaluations: (i) scalability under varying numbers of active LoRA tasks, (ii) robustness to length prediction errors under Gaussian noise and bucket-flipping perturbations, (iii) ablation of alternative penalty function designs, (iv) empirical validation of proportional-fairness convergence via per-adapter satisfaction ratios, and (v) comparison of adapter switching across methods and measurement of length prediction and scheduling overhead. Taken together, these additional evaluations further demonstrate FELPS's effectiveness in balancing efficiency and fairness.

## 6 Conclusion

In this paper, we propose FELPS, a fairness- and efficiency-aware scheduling framework tailored for the multi-LoRA LLM serving system. By integrating service-aware proportional-fair prioritization with an adjustable locality-aware penalty mechanism, FELPS achieves tunable trade-offs between fairness and efficiency. Extensive experiments under realistic *Conversation* and *Multi-task* workloads demonstrate that FELPS consistently outperforms all baselines across a wide range of system configurations, achieving a clearly superior trade-off between fairness and efficiency. These results establish FELPS as a practical solution for scalable and equitable multi-LoRA LLM serving.

## Acknowledgments and Disclosure of Funding

This work was sponsored by the National Natural Science Foundation of China (Nos. 62441229, 62672371 and 62472327) and the Innovative Research Group Project of Hubei Province (No. 2024AFA017). This work was also supported by the WHU-Kingsoft Joint Lab.

**Competing Interests.** The authors declare no competing financial interests relevant to this work.

## References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia, July 2018.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient llm scheduling by learning to rank. In *Advances in Neural Information Processing Systems*, volume 37, pages 59006–59029. Curran Associates, Inc., 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799, 2019.

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

Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 21 (1):2022–2023, 1984.

Shashwat Jaiswal, Shrikara Arun, Anjaly Parayil, Ankur Mallick, Spyros Mastorakis, Alind Khare, Renee St. Amant, Chetan Bansal, Victor Rühle, Chloi Alverti, and Josep Torrellas. Serving heterogeneous lora adapters in distributed llm inference systems. *arXiv preprint arXiv:2511.22880*, 2025.

Frank Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

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.

Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. Caraserve: Cpu-assisted and rank-aware lora serving for generative llm inference. *arXiv preprint arXiv:2401.11240*, 2024.

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

Xiao-Yang Liu, Guoxuan Wang, Hongyang Yang, and Daochen Zha. Fingpt: Democratizing internet-scale data for financial large language models. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

OpenAI. Gpt-4o. `https://openai.com/index/hello-gpt-4o/`, 2024.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4, 2023.

Nam Pham. Tiny-codes., 2023. URL `https://huggingface.co/datasets/nampdn-ai/tiny-codes`.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017.

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

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pages 965–988, 2024.

Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*. John Wiley & Sons, 2019.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

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

D Tse. Forward link multiuser diversity through proportional fair scheduling. *presentation at Bell Labs*, 1999.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dlora: Dynamically orchestrating requests and adapters for lora llm serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 911–927, 2024.

Yifei Xia, Fangcheng Fu, Wentao Zhang, Jiawei Jiang, and Bin Cui. Efficient multi-task llm quantization and serving for multiple lora adapters. *Advances in Neural Information Processing Systems*, 37:63686–63714, 2024.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6980–7008, Vienna, Austria, July 2025. ISBN 979-8-89176-256-5.

Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K Wong, Zhenguo Li, and Hengshuang Zhao. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *IEEE Robotics and Automation Letters*, 2024.

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

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*, pages 521–538, 2022.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online, July 2020.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Yilun Zhao, Zhenting Qi, Linyong Nan, Boyu Mi, Yixin Liu, Weijin Zou, Simeng Han, Ruizhe Chen, Xiangru Tang, Yumo Xu, Dragomir Radev, and Arman Cohan. QTSumm: Query-focused summarization over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1157–1172, Singapore, December 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. Pets: A unified framework for {Parameter-Efficient} transformers serving. In *2022 USENIX Annual Technical Conference*, pages 489–504, 2022.

# A Proof of the Proportional Fairness Guarantee

**Theorem A.1** (Proportional Fairness under SFP)**.** *Consider a work-conserving server with finite capacity $\mu_{\max}$ serving $N$ adapters with workload arrival rates $\omega_a = \lambda_a \bar{L}_a$. Under strict overload ($\Lambda = \sum_a \omega_a > \mu_{\max}$) and max-SFP scheduling in the fluid limit, the long-run service rates satisfy*

$$\lim_{t\to\infty} \frac{\bar{\mu}_i}{\bar{\mu}_j} = \frac{\omega_i}{\omega_j}, \quad \forall i, j \in \mathcal{B}, \tag{12}$$

*where $\mathcal{B} = \{a : \bar{\mu}_a < \omega_a\}$ is the bottleneck set and $\bar{\mu}_a$ is the long-run average service rate of adapter $a$.*

We prove the theorem in the fluid limit. All quantities are in the workload (token) domain.

### A.1 System Model and Assumptions

Let the server have finite capacity $\mu_{\max} > 0$ (tokens/sec). For each adapter $a \in \mathcal{A}$, $\omega_a = \lambda_a \bar{L}_a$ is the workload arrival rate, $Q_a(t)$ is the outstanding backlog, and $\mu_a(t)$ is the instantaneous service rate. The SFP priority is $\mathrm{SFP}_a(t) = W_a(t)/(\tilde{S}_a(t) + \varepsilon)$, where $W_a(t)$ is the cumulative waiting cost and $\tilde{S}_a(t)$ is the EMA-smoothed service history.

**Assumption A.2** (Strict Overload)**.** $\Lambda \triangleq \sum_{a\in\mathcal{A}} \omega_a > \mu_{\max}$.

**Assumption A.3** (Fluid Limit)**.** Under FSLLN scaling, the backlog evolves as $\dot{Q}_a(t) = \omega_a - \mu_a(t)$, and the server is work-conserving: $\sum_a \mu_a(t) = \mu_{\max}$ whenever $\sum_a Q_a(t) > 0$.

**Assumption A.4** (EMA Convergence)**.** In steady state, $\tilde{S}_a(t) \to \bar{\rho}_a \triangleq \bar{\mu}_a/\mu_{\max}$, where $\bar{\mu}_a = \lim_{t\to\infty} \frac{1}{t}\int_0^t \mu_a(s)\, ds$.

Within each adapter's queue, request is served in FCFS order. Define the *bottleneck set* of backlog adapters $\mathcal{B} = \{a \in \mathcal{A} : \bar{\mu}_a < \omega_a\}$. Under strict overload, $\mathcal{B} \neq \emptyset$. Adapters outside $\mathcal{B}$ ($\bar{\mu}_a \geq \omega_a$) correspond to sporadic low-frequency tasks whose queues are rapidly drained under SFP and do not contribute to sustained overload analysis.

### A.2 Waiting Cost Asymptotics

**Lemma A.5** (Quadratic Waiting Cost)**.** *For any $a \in \mathcal{B}$, $Q_a(t) = (\omega_a - \bar{\mu}_a)t + o(t)$, and*

$$W_a(t) = \frac{(\omega_a - \bar{\mu}_a)^2}{2\omega_a}\, t^2 + o(t^2). \tag{13}$$

*Proof.* Integrating the fluid ODE: $Q_a(t) = (\omega_a - \bar{\mu}_a)t + o(t)$. By FCFS, the oldest request still queued at time $t$ arrived at $s_a^*(t) = (\bar{\mu}_a/\omega_a)t + o(t)$. Summing waiting times over all continuously arriving work:

$$W_a(t) = \int_{s_a^*(t)}^{t} (t-s)\,\omega_a\, ds = \frac{\omega_a}{2}\big(t - s_a^*(t)\big)^2 = \frac{(\omega_a - \bar{\mu}_a)^2}{2\omega_a} t^2 + o(t^2). \tag{14}$$

□

### A.3 Priority Equalization via Contradiction

By Assumption A.4, for $a \in \mathcal{B}$ with $\bar{\mu}_a > 0$, the SFP priority satisfies

$$\mathrm{SFP}_a(t) = c_a t^2 + o(t^2), \qquad c_a \triangleq \frac{(\omega_a - \bar{\mu}_a)^2}{2\omega_a(\bar{\rho}_a + \varepsilon)}. \tag{15}$$

For the asymptotic analysis we take $\varepsilon \to 0$ (since $\bar{\rho}_a > 0$ in steady state), giving

$$c_a = \frac{(\omega_a - \bar{\mu}_a)^2}{2\omega_a \bar{\rho}_a} = \mu_{\max}\, g_{\omega_a}(\bar{\mu}_a), \quad g_\omega(\mu) \triangleq \frac{(\omega - \mu)^2}{2\omega\mu}. \tag{16}$$

Differentiating: $g'_\omega(\mu) = (\mu^2 - \omega^2)/(2\omega\mu^2) < 0$ for $0 < \mu < \omega$, so $g_\omega$ is strictly decreasing.

Suppose for contradiction that $c_i > c_j$ for some $i, j \in \mathcal{B}$. Then $\mathrm{SFP}_i(t) - \mathrm{SFP}_j(t) = (c_i - c_j)t^2 + o(t^2) \to \infty$, so after some finite $T$, adapter $j$ is permanently dominated in priority. Under max-SFP, $j$ receives zero service on $[T, \infty)$, forcing $\bar{\mu}_j = 0$. But if $\bar{\mu}_j = 0$, then $\bar{\rho}_j = 0$ and $\mathrm{SFP}_j(t) = W_j(t)/\varepsilon \to \infty$, contradiction. Hence all coefficients equalize: $c_a = \kappa$ for all $a \in \mathcal{B}$.

### A.4 Proportional-Fair Allocation

Introducing $x_a = \bar{\mu}_a/\omega_a \in (0, 1)$, the equalized coefficient condition becomes

$$\frac{(1 - x_a)^2}{2x_a} = \frac{\kappa}{\mu_{\max}}, \quad \forall a \in \mathcal{B}. \tag{17}$$

Since $h(x) = (1 - x)^2/(2x)$ is injective on $(0, 1)$, all $x_a = x$ are equal. By work conservation, the residual capacity for bottleneck adapters is $\mu_{\mathcal{B}} = \mu_{\max} - \sum_{a \notin \mathcal{B}} \omega_a$, giving $x = \mu_{\mathcal{B}} / \sum_{a \in \mathcal{B}} \omega_a$ and

$$\bar{\mu}_a = x\,\omega_a = \frac{\omega_a}{\sum_{b \in \mathcal{B}} \omega_b}\,\mu_{\mathcal{B}}, \quad a \in \mathcal{B}. \tag{18}$$

Equivalently, $\bar{\mu}_i/\bar{\mu}_j = \omega_i/\omega_j$ for all $i, j \in \mathcal{B}$, and likewise $\bar{\rho}_i/\bar{\rho}_j = \omega_i/\omega_j$. □

## B Theoretical Analysis of Locality-Aware Penalty

To rigorously quantify the impact of the dynamic penalty $\beta$ on service distribution, we extend our analysis to the *micro-scheduling cycle*. We demonstrate that the long-term service allocation is determined not only by the workload weight $\omega_a$ but also by the task's *locality* (or burstiness).

**Theorem B.1** (Penalty-Modulated Allocation Bound)**.** *Under penalty-weighted max-priority scheduling, the long-run service allocation of bottleneck adapters satisfies*

$$\frac{h(x_i)}{\beta_{\textit{eff},i}} = \frac{h(x_j)}{\beta_{\textit{eff},j}}, \quad \forall i, j \in \mathcal{B}, \qquad h(x) = \frac{(1 - x)^2}{2x}, \tag{19}$$

*where $x_a = \bar{\mu}_a/\omega_a$. Consequently, the allocation ratio is multiplicatively bounded:*

$$\frac{\bar{\rho}_i}{\bar{\rho}_j} = \frac{\omega_i}{\omega_j} \cdot \frac{\beta_{\textit{eff},j}}{\beta_{\textit{eff},i}} \cdot \gamma_{ij}, \qquad \gamma_{ij} \in \big((1 - \delta)^2,\ (1 - \delta)^{-2}\big), \tag{20}$$

*where $\delta = \max_{a \in \mathcal{B}} x_a < 1$. In the limit $\delta \to 0$, $\gamma_{ij} \to 1$ and the ratio converges to the penalty-proportional form.*

*Proof.* We extend the contradiction argument of Appendix A to account for the time-varying penalty $\beta_a(t)$.

### B.1 Duty Cycle in the Micro-Scheduling Limit Cycle

In the steady state, the system reaches a limit cycle where each adapter $a$ alternates between two phases: a *loaded phase* of duration $\tau_{\text{run},a}$ (adapter resident in GPU, $\beta_a = 1$) and a *waiting phase* of duration $\tau_{\text{wait},a}$ (adapter evicted, $\beta_a = \beta_{\text{high},a}$). This cycle repeats with period $T_a = \tau_{\text{run},a} + \tau_{\text{wait},a}$. The duty cycle is $\mathcal{D}_a = \tau_{\text{run},a}/T_a \in (0, 1)$.

*Relationship between $\mathcal{D}_a$ and $\bar{\mu}_a$.* During the loaded phase, adapter $a$ competes for service with other loaded adapters. Let $\mu_a^{\text{on}}$ denote its service rate when loaded. Then the long-run average satisfies $\bar{\mu}_a = \mathcal{D}_a \cdot \mu_a^{\text{on}}$. In particular, $\bar{\mu}_a \le \mathcal{D}_a \cdot \mu_{\max}$. For bottleneck adapters with $\bar{\mu}_a < \omega_a$, the backlog grows, and the system must periodically reload them; hence $\mathcal{D}_a > 0$. The exact value of $\mathcal{D}_a$ is endogenously determined by the equilibrium, but—crucially—it converges to a constant in steady state because the system trajectories enter a stable limit cycle. We therefore treat $\mathcal{D}_a$ as a steady-state constant and solve for the self-consistent equilibrium allocation $\{\bar{\mu}_a, \mathcal{D}_a\}$.

### B.2 Asymptotic Penalty-Weighted Priority

The scheduling priority is $P_a(t) = \mathrm{SFP}_a(t)/\beta_a(t)$. From Appendix A, $\mathrm{SFP}_a(t) = c_a t^2 + o(t^2)$ where $c_a = (\omega_a - \bar{\mu}_a)^2/(2\omega_a\bar{\rho}_a)$ (taking $\varepsilon \to 0$ in steady state). Since the SFP grows as $\Theta(t^2)$ on

the macro timescale while $\beta_a(t)$ oscillates with bounded period $T_a$, we can separate the timescales: for any macro time $T \gg T_a$, the number of cycles on $[0, T]$ is $T/T_a \to \infty$, so the time-averaged effect of $1/\beta_a(t)$ converges almost surely:

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \frac{1}{\beta_a(t)}\, dt = \frac{1}{T_a}\Big(\int_0^{\tau_{\text{run},a}} 1\, dt + \int_0^{\tau_{\text{wait},a}} \frac{1}{\beta_{\text{high},a}}\, dt\Big) = \mathcal{D}_a + \frac{1-\mathcal{D}_a}{\beta_{\text{high},a}} \triangleq \frac{1}{\beta_{\text{eff},a}}. \quad (21)$$

Solving for $\beta_{\text{eff},a}$ recovers $\beta_{\text{eff},a} = \beta_{\text{high},a}/(1 + \mathcal{D}_a(\beta_{\text{high},a} - 1))$, matching the definition in Section 4.1.

Now consider the asymptotic coefficient of $P_a(t)$. At a scheduling decision instant $t$, $P_a(t) = (c_a t^2 + o(t^2))/\beta_a(t)$. Over any macro interval $[T, T+\Delta]$, the max-priority scheduler selects adapters based on the envelope of these values. Since $\beta_a(t)$ is piecewise constant, the temporal selection frequencies are governed by the *effective coefficient* $c_a^{\text{eff}} = c_a/\beta_{\text{eff},a}$. Formally, for two bottleneck adapters $i, j$, if $c_i^{\text{eff}} > c_j^{\text{eff}}$, then there exists $T$ such that for all $t \geq T$,

$$\frac{\text{SFP}_i(t)}{\beta_i(t)} - \frac{\text{SFP}_j(t)}{\beta_j(t)} > 0 \quad \text{for a strictly positive fraction of each cycle,} \quad (22)$$

and the long-run selection ratio satisfies $\bar{\mu}_j \to 0$. This reproduces the contradiction mechanism of Appendix A.

### B.3 Modified Equilibrium

Equalizing effective coefficients gives $c_i/\beta_{\text{eff},i} = c_j/\beta_{\text{eff},j} = \kappa$ for all $i, j \in \mathcal{B}$. Substituting $c_a = \mu_{\max} \cdot g_{\omega_a}(\bar{\mu}_a)$ with $g_\omega(\mu) = (\omega - \mu)^2/(2\omega\mu)$ from Appendix A:

$$\frac{(\omega_a - \bar{\mu}_a)^2}{2\omega_a \bar{\mu}_a} \cdot \frac{1}{\beta_{\text{eff},a}} = \frac{\kappa}{\mu_{\max}}, \qquad a \in \mathcal{B}. \quad (23)$$

Let $x_a = \bar{\mu}_a/\omega_a \in (0, 1)$ and define $h(x) = (1-x)^2/(2x)$. Then Eq. (23) becomes

$$\frac{h(x_a)}{\beta_{\text{eff},a}} = \frac{\kappa}{\mu_{\max}}, \qquad a \in \mathcal{B}. \quad (24)$$

**Bounding the allocation ratio.** For any two bottleneck adapters $i, j$, the equalization condition $h(x_i)/\beta_{\text{eff},i} = h(x_j)/\beta_{\text{eff},j}$ gives

$$\frac{x_i}{x_j} = \frac{\beta_{\text{eff},j}}{\beta_{\text{eff},i}} \cdot \frac{(1-x_i)^2}{(1-x_j)^2} \triangleq \frac{\beta_{\text{eff},j}}{\beta_{\text{eff},i}} \cdot \gamma_{ij}. \quad (25)$$

We now bound $\gamma_{ij}$ in terms of the penalty ratio rather than the load regime. Without loss of generality, let $r = \beta_{\text{eff},i}/\beta_{\text{eff},j} \geq 1$, so $h(x_i) = r \cdot h(x_j)$ and $x_i < x_j$. Substituting $u = x_i/x_j \in (0, 1)$ into the equalization:

$$\frac{(1-ux_j)^2}{u} = r \cdot (1-x_j)^2. \quad (26)$$

Define $f(u) = (1-ux_j)^2 - ur(1-x_j)^2$. At $u = 1/r$: $f(1/r) = (1-x_j/r)^2 - (1-x_j)^2 > 0$ (since $r > 1$). At $u = 1$: $f(1) = (1-x_j)^2(1-r) < 0$. Moreover, $f'(u) = -2x_j(1-ux_j) - r(1-x_j)^2 < 0$ for all $u \in (0, 1)$ (since $1 - ux_j > 0$), so $f$ is strictly decreasing on $(0, 1)$. Hence there exists a *unique* root $u^* \in (1/r, 1)$, which is the equilibrium $x_i/x_j$. Consequently,

$$\gamma_{ij} = r \cdot u^* \in \big(1,\ r\big). \quad (27)$$

When $\beta_{\text{eff},i} < \beta_{\text{eff},j}$, swapping the roles yields $\gamma_{ji} = 1/\gamma_{ij} \in (1,\ \beta_{\text{eff},j}/\beta_{\text{eff},i})$, i.e., $\gamma_{ij} \in (\beta_{\text{eff},i}/\beta_{\text{eff},j},\ 1)$. So for any pair $i, j$, letting $B_{ij} = \max(\beta_{\text{eff},i}/\beta_{\text{eff},j}, \beta_{\text{eff},j}/\beta_{\text{eff},i})$ we have $\gamma_{ij} \in (B_{ij}^{-1}, B_{ij})$. Taking $B = \max_{p,q\in\mathcal{B}} B_{pq}$ yields the uniform bound $\gamma_{ij} \in (B^{-1}, B)$ for all $i, j$.

The range of $B$ is inherently limited: $\beta_{\text{eff},a} \in [1, \beta_{\text{high},a}]$ and $\beta_{\text{high},a} = 1 + \lambda \log_2(1 + \text{rank}_a/32)$, so $B$ is bounded by the maximum configured penalty ratio across all adapters. For typical configurations ($\lambda = 1$, ranks $\leq 64$), $B \leq 1 + \log_2(3) \approx 2.6$, yielding $\gamma_{ij} \in (0.38, 2.6)$. Substituting into Eq. (25):

$$\frac{\bar{\rho}_i}{\bar{\rho}_j} = \frac{\omega_i}{\omega_j} \cdot \frac{\beta_{\text{eff},j}}{\beta_{\text{eff},i}} \cdot \gamma_{ij}, \qquad \gamma_{ij} \in (B^{-1},\ B). \quad (28)$$

### B.4 Starvation Prevention Guarantee

A critical consequence of Eq. (24) is that $x_a > 0$ for every bottleneck adapter. Since $\beta_{\text{eff},a} < \infty$ (the harmonic mean of a bounded function is bounded) and $h(x_a) \to \infty$ as $x_a \to 0^+$, there is a unique $x_a \in (0, 1)$ solving the equation for any finite $\beta_{\text{eff},a}$. This guarantees $\bar{\mu}_a > 0$ for all $a \in \mathcal{B}$: the penalty *modulates* the allocation but never reduces it to zero. No adapter is permanently starved, confirming that the locality-aware penalty achieves its design goal of suppressing switching overhead while preserving proportional fairness in the long run. □

## C Experiment Details

### C.1 Hardware Environment

To evaluate the system's performance under different resource constraints, we conduct experiments on two distinct GPU configurations. Specifically, we deploy **Llama-2-7B** on a consumer-grade **NVIDIA RTX 4090** (24GB), while the larger **Llama-2-13B** model is deployed on a professional-grade **NVIDIA RTX A6000** (48GB) to accommodate its higher memory footprint. The detailed specifications of these two GPU platforms are listed in Table 4. Our code is available at `https://github.com/Kydd1/FELPS`.

Table 4: Detailed specifications of the GPUs used in our experiments.

| **Specification** | **NVIDIA RTX 4090** | **NVIDIA RTX A6000** |
|---|---|---|
| Architecture | Ada Lovelace | Ampere |
| GPU Memory | 24 GB GDDR6X | 48 GB GDDR6 |
| Memory Bandwidth | 1,008 GB/s | 768 GB/s |
| CUDA Cores | 16,384 | 10,752 |
| Tensor Cores | 512 (4th Gen) | 336 (3rd Gen) |
| TDP | 450 W | 300 W |
| **Deployed Model** | **Llama-2-7B** | **Llama-2-13B** |
| Memory Pool Size | 10,000 | 20,000 |

### C.2 Dataset and Workload

For the *Multi-task* workload, we constructed a diverse benchmark comprising 10 distinct downstream tasks sampled from nine publicly available datasets. These tasks cover a wide spectrum of domains, including coding, creative writing, summarization, translation, and reasoning. Crucially, they exhibit highly heterogeneous input and output length distributions, ranging from context-heavy summarization to generation-heavy writing tasks. The detailed characteristics of each task template are summarized in Table 5.

**Workload Configuration.** To evaluate performance under appropriate memory pressure for each hardware configuration, we scale the number of active LoRA tasks based on the GPU capacity. Specifically, for the **Llama-2-7B** experiments on the RTX 4090 (24GB), we configure the workload with **20 tasks**; for the **Llama-2-13B** experiments on the RTX A6000 (48GB), we scale the workload up to **50 tasks** to simulate higher concurrency and resource contention.

**Traffic Generation.** To simulate realistic request dynamics, we model the arrival intervals using a **Gamma process** with the Coefficient of Variation (CV) set to 1. Furthermore, task popularity is sampled according to a **Zipf distribution** to replicate real-world long-tail skewness. The Zipf skewness parameter $\alpha$ is calibrated within the range of $(1.0, 1.1]$, varying slightly depending on the hardware environment to maintain consistent stress levels across different setups. The workload execution is sustained for a duration of **5 minutes**, and the average request rates are fixed at 8 req/s for Llama-2-7B and 5 req/s for Llama-2-13B.

**Conversation Workload.** For the *Conversation* workload, we utilize the **Chatbot Arena Conversations** dataset [Zheng et al., 2023], a large-scale collection containing 33,000 real-world human-bot dialogue instances generated by 20 different LLM-based chatbots. To adapt this for multi-LoRA serving, we map the unique source LLM identities to distinct LoRA adapters. Consistent with

Table 5: Detailed characteristics of the 10 tasks used in the Multi-task workload.

| Task Category | Dataset | Input Len. | Output Len. |
|---|---|---|---|
| Coding | tiny-codes [Pham, 2023] | 65 | 512 |
| | KodCode-V1 [Xu et al., 2025] | 220 | 210 |
| Writing | WritingPrompts [Fan et al., 2018] | 35 | 750 |
| | Alpaca-GPT4 [Peng et al., 2023] | 27 | 163 |
| Summary | CNN_DailyMail [See et al., 2017] | 930 | 68 |
| Query-Summ | QTsumm [Zhao et al., 2023] | 29 | 102 |
| Math | GSM8k [Cobbe et al., 2021] | 65 | 129 |
| Translation | Opus-en-ja [Zhang et al., 2020] | 75 | 154 |
| | Opus-en-zh [Zhang et al., 2020] | 78 | 136 |
| Multiple Choice | MMLU [Hendrycks et al., 2021] | 400 | 2 |

our hardware-aware scaling, we configure **20 LoRA tasks** for the Llama-2-7B experiments (RTX 4090) and scale up to **40 LoRA tasks** for the Llama-2-13B experiments (RTX A6000). To ensure compatibility with the context window, we filter instances to strictly keep both input and output lengths within the range of $[0, 1024]$ tokens. Unlike the skewed multi-task workload, requests here are sampled **uniformly** across tasks as the dataset inherently contains realistic request timestamps. The workload execution is sustained for a duration of **5 minutes**, and the average request rates are fixed at 8 req/s for Llama-2-7B and 5 req/s for Llama-2-13B.

### C.3 Baseline Descriptions

**FCFS (First-Come-First-Served) [Silberschatz et al., 2019].** This is the standard scheduling policy used in many serving systems. FCFS processes requests strictly in the order of their arrival timestamps. While it guarantees fairness in terms of waiting time, it lacks awareness of LoRA adapter locality. Consequently, interleaved requests for different adapters can trigger frequent adapter swapping, leading to severe memory contention and throughput degradation.

**AC (Adapter Clustering) [Sheng et al., 2023].** Originally proposed in S-LoRA, AC is a throughput-oriented strategy designed to minimize adapter switching overhead. It prioritizes batching requests that share the same LoRA adapter. To prevent memory overflow from too many active adapters, AC enforces a limit on the number of concurrently loaded adapters. In our experiments, we set this concurrent adapter limit to 5. Requests targeting adapters outside this active set must wait until an active adapter slot becomes available, which maximizes reuse but can lead to starvation for less popular tasks.

**Inlaid (LoRA-Inlaid) [Xia et al., 2024].** Inlaid is a latency-optimized scheduler that adapts the Shortest Remaining Time First (SRTF) principle to LoRA serving. It estimates the remaining processing time of requests and prioritizes those that can finish quickest to minimize average job completion time. Similar to AC, it employs an admission control mechanism to limit memory usage. To mitigate the inherent starvation issue of SRTF, Inlaid includes a starvation prevention mechanism. Requests waiting longer than this duration are promoted to a high-priority queue to ensure they are eventually served. In our experiments, the concurrent adapter limit is set to 5, and the starvation threshold is set to 60 seconds.

**VTC (Virtual Token Counter) [Sheng et al., 2024].** VTC represents the state-of-the-art fairness-centric scheduler derived from fair queuing theory. It treats each LoRA task (or client) as a distinct user and tracks their accumulated service usage via a virtual token counter. At each scheduling step, VTC strictly prioritizes the task with the lowest accumulated usage (i.e., the "poorest" task) to enforce equal resource distribution. While this maximizes absolute fairness, it often disregards locality, causing frequent context switching and memory fragmentation that harms overall system throughput.

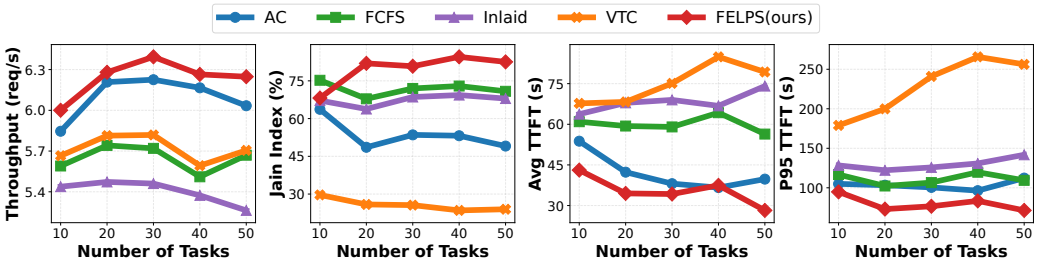



Figure 5: Impact of number of LoRA tasks.

# D Additional Evaluation

## D.1 Scalability with Number of Tasks

Figure 5 investigates the scalability of FELPS by varying the number of active LoRA tasks from 10 to 50. As the number of tasks increases, memory contention intensifies, challenging the scheduler's ability to balance resources.

**Throughput & Efficiency:** FELPS consistently outperforms all baselines in throughput. It matches and slightly exceeds the efficiency-oriented AC, demonstrating that our locality-aware penalty effectively minimizes switching overheads even under high concurrency.

**Fairness:** A critical divergence is observed in the Jain Index. While AC's fairness degrades significantly (dropping below 50%) as tasks increase due to its aggressive locking of active adapters, FELPS maintains a superior fairness score ($> 80\%$), surpassing even the fairness-oriented FCFS.

**Latency:** In terms of TTFT, FELPS achieves the lowest average and, more importantly, the lowest P95 latency. Unlike VTC, which suffers from exploding latency metrics, FELPS ensures that no specific task experiences starvation, validating the effectiveness of our service-aware priority mechanism. In summary, FELPS is the only method that scales robustly, simultaneously delivering high efficiency and strict fairness without compromising tail latency.

## D.2 Length Prediction Robustness

To evaluate the sensitivity of FELPS to length prediction errors, we conduct systematic robustness experiments on Llama-2-7B under the *Multi-task* workload. We inject two types of controlled noise into the predictor output: (1) **Gaussian noise** with varying standard deviation $\sigma$, and (2) **bucket random flipping** with probability $p$. Table 6 reports the results. Even under massive perturbations ($\sigma = 0.5$ or $p = 0.3$), FELPS degrades gracefully: throughput remains stable ($> 6.2$ req/s) and Jain's index drops by less than 10 pp. This robustness stems from SFP's reliance on cumulative relative sequence costs for priority ranking: prediction errors only marginally perturb the relative ordering among adapters.

Table 6: Robustness of FELPS to length prediction errors (Llama-2-7B).

| Type | Level | Pred. MAE | Pred. AE | Tput | Avg TTFT | P95 TTFT | Jain (%) |
|---|---|---|---|---|---|---|---|
| — | FELPS (clean) | 104 (31.8%) | −53 (−16.2%) | 6.28 | 34.48 | 73.05 | 81.95 |
| Gaussian noise | $\sigma = 0.1$ | 104 (31.8%) | −53 (−16.2%) | 6.23 | 38.45 | 81.35 | 74.13 |
| | $\sigma = 0.3$ | 107 (32.7%) | −48 (−14.7%) | 6.27 | 37.81 | 78.68 | 75.73 |
| | $\sigma = 0.5$ | 108 (33.0%) | −38 (−11.6%) | 6.30 | 36.72 | 79.39 | 75.11 |
| Bucket flipping | $p = 0.05$ | 105 (32.1%) | −50 (−15.3%) | 6.21 | 37.72 | 78.63 | 73.29 |
| | $p = 0.1$ | 106 (32.4%) | −47 (−14.4%) | 6.24 | 37.67 | 80.95 | 73.83 |
| | $p = 0.3$ | 108 (33.0%) | −38 (−11.6%) | 6.27 | 38.02 | 81.62 | 72.91 |

### D.3 Penalty Function Ablation

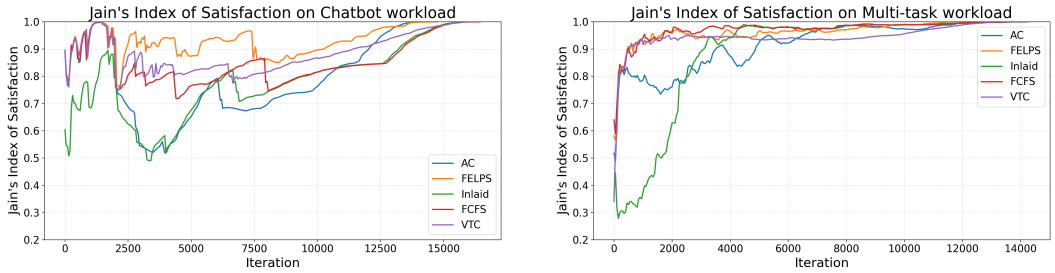
We compare four penalty designs under the same $\lambda$ value on Llama-2-7B with the *Multi-task* workload. Table 7 reports the results. SFP-only ($\lambda = 0$) or a uniform rank-blind penalty ($\beta = 1 + \lambda$) severely impairs throughput. A linear penalty ($1 + \lambda \cdot r/32$) slightly improves throughput but fatally starves large-rank LoRAs, dropping Jain's index to 74.99%. Our log-scaled penalty achieves the optimal balance, consistent with the sub-linear principle: the marginal switching overhead decreases with adapter size due to amortized fixed costs (e.g., PCIe latency).

Table 7: Comparison of penalty function designs (Llama-2-7B, Multi-task).

| Penalty Function | Tput | Avg TTFT | P95 TTFT | Jain (%) |
|---|---|---|---|---|
| SFP-only ($\lambda = 0$) | 5.60 | 61.42 | 113.54 | 91.55 |
| Uniform ($\beta = 1 + \lambda$) | 5.83 | 43.76 | 102.10 | 80.79 |
| Linear ($1 + \lambda \cdot r/32$) | 6.33 | 38.39 | 78.66 | 74.99 |
| Log ($1 + \lambda \cdot \log_2(1 + r/32)$) | 6.28 | 34.48 | 73.05 | 81.95 |

### D.4 Satisfaction Fairness Convergence

To empirically validate the proportional-fairness guarantee (Eq. 8), we measure the **Jain's index of per-adapter satisfaction ratios** over time, where $\text{sat}_a(t) = \text{cumulative tokens served}_a(t) \,/\, \text{cumulative tokens requested}_a(t)$. A Jain's index close to 1 indicates that each adapter's service is strictly proportional to its demand. Figure 6 shows the satisfaction fairness trajectories on Llama-2-7B under both workloads. FELPS converges to $J \approx 1.0$ as rapidly as the fairness-oriented FCFS and VTC, while simultaneously matching the throughput of the efficiency-oriented AC. In contrast, AC and Inlaid exhibit significantly larger fairness oscillations, confirming that their efficiency gains come at the cost of disproportionate service.




(a) Chatbot workload.

(b) Multi-task workload.

Figure 6: Jain's index of per-adapter satisfaction ratios over time (Llama-2-7B).

### D.5 System Overhead Analysis

We profile adapter switching and weight transfers, length prediction, and scheduling decisions to characterize the system costs of FELPS. All measurements in this subsection use Llama-2-7B.

**Adapter Switching and Weight Transfers.** Table 8 reports adapter switches per thousand requests on the *Conversation* and *Multi-task* workloads. FELPS has the lowest switching frequency among the evaluated methods, with approximately one-third as many switches as FCFS. For both AC and Inlaid, the concurrent adapter limit is set to 5. The total adapter weight-copy time for FELPS is below 0.5 s over a complete profiling run. Beyond weight-copy time, our locality-aware design targets memory contention between adapter weights and KV caches, which affects the batch capacity available for serving requests.

Table 8: Adapter switches per thousand requests. Lower is better.

| Scheduler | Conversation | Multi-task |
|---|---|---|
| FELPS | **60.4** | **94.6** |
| AC | 63.8 | 100.0 |
| Inlaid | 150.0 | 199.6 |
| VTC | 162.1 | 235.4 |
| FCFS | 190.4 | 271.2 |

**Length Predictor Overhead.** On the *Multi-task* workload, the OPT-125M length predictor takes 4.56 ms per request on average, with a P95 latency of 6.85 ms. Its static memory footprint is 479 MB, and its peak memory usage is 2.04 GB (Table 9). Prediction latency is small relative to the end-to-end serving latency in our evaluated workloads. The peak memory footprint only briefly affects serving performance, for several milliseconds on the longest input requests.

**Scheduling Overhead.** The `generate_new_batch` decision takes approximately 0.2 ms on average, with a P95 latency of 0.77 ms. Cumulative scheduling decision time accounts for 0.2–0.3% of the experiment duration, indicating a small CPU-side decision cost in the profiled runs.

Table 9: Measured length predictor and scheduler overhead. Prediction latency is measured per request; scheduling latency is measured per batch-construction decision.

| Component | Metric | Value |
|---|---|---|
| Length predictor | Mean latency | 4.56 ms |
| | P95 latency | 6.85 ms |
| | Static memory | 479 MB |
| | Peak memory | 2.04 GB |
| Scheduler | Mean decision time | $\approx$0.2 ms |
| | P95 decision time | 0.77 ms |
| | Total time / experiment duration | 0.2–0.3% |

## NeurIPS Paper Checklist

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 abstract and introduction accurately state the paper's contributions, with theoretical and experimental support.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: Section 4.3 Discussion lists three limitations: length predictor, single-GPU scope, and $\lambda$ calibration.

   Guidelines:

   - The answer [N/A] 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: [Yes]

Justification: Appendices A and B contain complete proofs with explicitly stated assumptions; proof sketches are provided in the main body.

Guidelines:

- The answer [N/A] 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: Section 5.1 and Appendix C contain full experimental setup details; a public code repository is provided.

   Guidelines:

   - The answer [N/A] 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: A public code repository is provided at `https://github.com/Kydd1/FELPS` (Appendix C).

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: Section 5.1 and Appendix C contain detailed experimental setup descriptions.

Guidelines:

- The answer [N/A] 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: [Yes]

Justification: We evaluate deterministic metrics under fixed workload traces. There is no repeated model training.

Guidelines:

- The answer [N/A] 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: Appendix C lists GPU specifications, memory pool sizes, and run durations.

   Guidelines:

   - The answer [N/A] 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: The research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] 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: [N/A]

    Justification: There is no societal impact of the work performed. Fairness discussed in this paper is about serving proportionally to each task, regardless of impacting special groups.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No new models, datasets, or generative artifacts are released.

    Guidelines:

    - The answer [N/A] 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 assets are cited with their original publications.

    Guidelines:

    - The answer [N/A] 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: The code is provided as a public open-source repository with documentation.

    Guidelines:

    - The answer [N/A] 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: [N/A]

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

    Guidelines:

    - The answer [N/A] 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: [N/A]

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

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

Answer: [N/A]

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

Guidelines:

- The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.