Justitia: Fair and Efficient Scheduling of Task-parallel LLM Agents with Selective Pampering

Mingyan Yang **Guanjie Wang** **Manqi Luo** **Yifei Liu** **Chen Chen***
Han Zhao **Yu Feng** **Quan Chen** **Minyi Guo**

Shanghai Jiao Tong University
{yangmingyan, irfnfnkemed, lmq20041203, nephrencake, chen-chen}@sjtu.edu.cn
zhao-han@cs.sjtu.edu.cn, y-feng@sjtu.edu.cn
{chen-quan, guo-my}@cs.sjtu.edu.cn

Abstract

LLM agents, which often comprise parallel inference tasks, are commonly adopted to solve real-world problems. When serving such task-parallel LLM agents in shared GPU servers, the scheduler is expected to attain fast agent completion with guaranteed worst-case performance. For that objective, our insight is to selectively pamper agents based on their completion order under idealized fair-sharing. We design *Justitia*, a fair and also efficient scheduler for task-parallel LLM agents. Noticing that memory is prevalently a bottleneck in LLM serving, *Justitia* quantifies the true agent cost in a memory-centric manner. It also adopts a light-weight yet accurate method to predict agent costs. Finally, *Justitia* adopts a virtual-time based fair queuing algorithm to improve the overall performance with guaranteed worst-case delay. We have implemented *Justitia* atop vLLM, and experimental results involving diverse agents show that it can substantially enhance the scheduling efficiency with fairness preserved.

1 Introduction

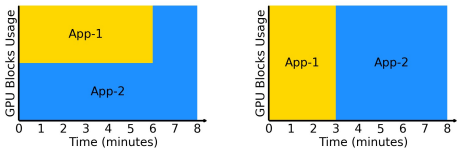
The recent years have witnessed the boom of large language models (LLMs) [1] in revolutionizing various fields [2, 3, 4]. In particular, it has become increasingly popular to address realistic problems with *LLM agents*, which usually contain *a set of parallel inference tasks* for better quality performance, as supported by the *Kimi Agent Swarm* mode [5]. For example, to summarize a very large file, multiple parallel inferences would be launched, each processing one file partition [6]; besides, to solve a complex mathematical problem, multiple searching directions need to be explored concurrently, until a satisfiable answer is obtained [7]. Moreover, such LLM agents are prevalently served in *shared GPU servers*, where users usually expect to get the final output as early as possible [6, 8], without being remarkably delayed by competitors [9, 10].

However, existing LLM serving systems fail to behave well in both efficiency and fairness for task-parallel LLM agents. For example, mainstream LLM frameworks [11, 12] schedule LLM inference requests in a *First-Come-First-Serve* (FCFS) manner, being inefficient due to *head-of-line-blocking*. Meanwhile, while prioritizing short jobs [13, 14] can improve efficiency, it however incurs starvation. To ensure fairness, the VTC scheduler [9] is proposed to fairly allocate the serving resources among different users. However, this method forces each agent to only use its fair share, resulting in postponed completion.

*Corresponding author.

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

To attain efficient and also fair serving, our insight is that LLM agents shall be served in a saturated manner following the fair completion order, which we call *selective agent pampering*. We first note that, regarding fairness, users primarily care about the *long-term* fairness (i.e., with guaranteed fast completion) instead of *short-term* fairness (i.e., always allocated an equal share of the total resources). In that sense, as shown in Fig. 1, when two agents are competing with each other, serving them, not *concurrently with the fair share*, but *sequentially in the fair completion order with all the resources*, can reduce the average completion time, with no agent actually delayed.



(a) Fair Sharing (b) Selective Pampering

Figure 1: Given two competing LLM agents, serving them sequentially following their fair completion order, or *selective pampering*, can reduce the average completion time with no per-agent delay.

Yet, it is a non-trivial task to apply the above insight into practice, which requires estimating the service cost of LLM agents and also calculating their fair completion order (i.e., the completion order under idealized fair sharing). Specifically, LLM inferences consume both compute and memory (i.e., KV cache) resources on GPU servers, rendering it difficult to precisely quantify an agent's true service cost. Meanwhile, the overall service cost of an LLM agent needs to be predicted upon its arrival; such prediction shall be made accurate and also efficient. Moreover, upon the arrival or completion of any agent, we need to work out the expected completion order of all the pending agents under a fair scheduler; such decisions shall be made efficiently (without frequent refreshing later) and also of high quality (yielding fast completion with worst-case guarantee).

In this paper, we propose *Justitia*, a fair and efficient scheduler for task-parallel LLM agents. Identifying KV-cache capacity as a dominant scheduling constraint in concurrent LLM serving, we introduce *memory-centric cost modeling*, jointly considering memory space and time, to replace traditional compute-centric approaches [9]. For cost prediction, *Justitia* employs a lightweight and accurate Multi-Layer Perceptron leveraging agent-specific demand stability. To determine queuing order, we adapt virtual-time-based *fair queuing* [15, 16], which efficiently calculates the completion order under an idealized fair scheduler in one shot, independent of future arrivals. Finally, we theoretically prove that the maximal delay any agent encounters under *Justitia*, relative to the idealized fair scheduler, is strictly bounded.

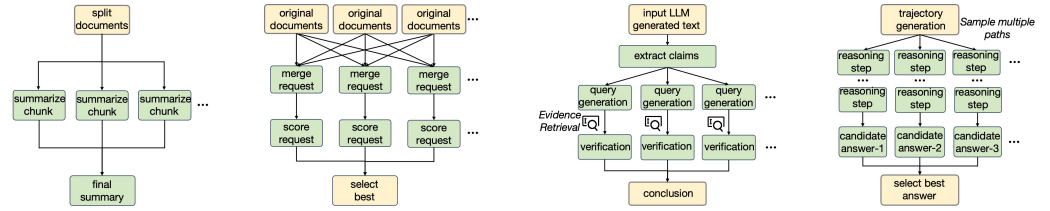
We have implemented *Justitia* above vLLM [11], a mainstream LLM serving framework. Given a set of diverse LLM agents, *Justitia* can remarkably improve the average completion time without remarkably delaying any agent. Specifically, compared to the state-of-the-art fair scheduler, VTC, our *Justitia* scheduler can reduce the average agent completion time by 57.5%, with only 8% of agents delayed (their average delay scale is less than 10%). Moreover, further studies confirm that it is indeed indispensable to adopt memory-centric service cost modeling and MLP-based demand prediction for LLM agents; besides, the scheduling overhead of *Justitia* is also negligible.

2 Background and Motivation

2.1 Task-parallel LLM Agents

The emergence of task-parallel LLM agents. Large language models (LLMs) excel in text processing [17], code generation [3], and planning [18]. To solve complex real-world problems, a growing trend involves deploying *LLM agents* where multiple inferences collaborate [6, 4].

For example, as shown in Fig. 2a, since the context window size of an LLM inference is limited (e.g., 10M tokens [19]), to summarize a large text file, multiple parallel inferences, each processing a slice of the original file, need to be launched and finally have their results merged [6]. Similarly, in Fig. 2b, to improve the merging quality of multiple documents, several merging inferences are submitted in parallel, each followed by a scoring inference [20]. In Fig. 2c, to address the *hallucination* problem [21], each claim within the LLM-generated output needs to be additionally verified by a dedicated LLM inference (e.g., with the *FacTool* framework [22]). In Fig. 2d, to solve complex



(a) MapReduce Summary (b) Doc Merging (c) Fact Verification (d) Self Consistency

Figure 2: Examples of task-parallel LLM agents.

mathematical problems, the self-consistency (SC) agent [23] expands multiple reasoning trajectories and applies majority voting to find the best answer.

We define these workloads as *task-parallel LLM agents*, collections of parallel inference tasks that must all complete before delivering the final output. As an increasingly dominant workload in LLM serving clusters, they are the primary focus of this paper.

Scheduling objective for task-parallel LLM agents. The primary performance metric for an LLM agent is its *end-to-end execution time* [6]. However, when multiple agents from different users compete for limited GPU resources, clusters must also ensure service fairness to prevent issues like *head-of-line-blocking* or *starvation* [9]. Consequently, *efficiency* and *fairness* constitute the two critical scheduling objectives for serving task-parallel LLM agents.

2.2 Related Works and Their Limitations

Regarding LLM inference serving, a series of scheduling methods have been proposed recently. Here we revisit such scheduling practices and summarize their limitations.

Due to the non-deterministic output length of LLM inferences, mainstream frameworks (vLLM [11], SGLang [12]) and agent-level schedulers (Parrot [6], Ayo [8]) commonly adopt First-Come-First-Serve (FCFS). However, they suffer the head-of-line-blocking problem, meaning that a large task would impede the execution of those behind it.

Given the inefficiency of FCFS schedulers, some other schedulers are designed to be *efficiency-centric*. For example, to tackle head-of-line-blocking, FastServe [24] employs a *Multi-Level-Feedback-Queue* (MLFQ) for scheduling, yet it still compromises the end-to-end execution latency due to the periodical execution mode. Recently, some prediction-based methods propose to approximate *Shortest-Job-First* (SJF) in scheduling [25, 13, 14], yielding near-optimal efficiency. However, such efficiency-centric schedulers may starve large agents, failing to provide service guarantee.

To guarantee the worst-case performance, some *fairness-centric* schedulers are also proposed. For example, Sheng *et al.* proposed a fair scheduler called *Virtual Token Counter* (VTC) [9], which tracks the services received for each tenant and prioritizes the ones receiving the least services. Another work, *FairServe* [10], uses a similar scheduling method to avoid cross-tenant interference. However, by enforcing each task-parallel agent to only use its fair share, the end-to-end execution time, which users truly care about, would be delayed compared to that with monopolized resources.

In summary, existing LLM scheduling methods fail to balance agent-level efficiency and fairness, compromising user experience. To bridge this gap, we propose a scheduler for task-parallel LLM agents that optimizes efficiency by minimizing average completion time, while guaranteeing worst-case service degradation remains bounded by an equal resource share.

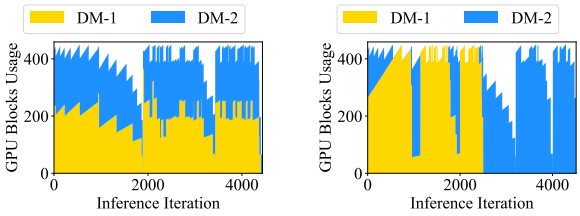
2.3 Insight: Reconciling Fairness and Efficiency with Selective Agent Pampering

***Long-term fairness*: rationale and definition.** While efficiency and fairness are appealing objectives, they are however conflicting with each other. We note that fairness-centric schedulers [9, 10] are inefficient essentially because they seek to fairly share the resources at each instant moment, i.e., stick to *instantaneous fairness*. However, in expecting service fairness, users primarily desire a service guarantee on the worst-case completion time at the agent level. In that sense, long-term fairness (i.e., guaranteeing agent completion time) would suffice for users; a typical fairness definition

in that regard is *finish-time fairness* [26], which relates to *the ratio of the running time in a shared cluster with N tenants to running alone in a $\frac{1}{N}$ cluster*. We thus materialize our fairness objective as *long-term finish-time fairness*, which allows for more flexibility to balance between both sides.

***Selective pampering*: a powerful methodology to optimize efficiency while preserving long-term fairness.** Instantaneous fair sharing limits task-parallel LLM agents to average resource shares. Instead, *selective pampering* prioritizes agents sequentially based on *their relative completion order under fair sharing*, granting them unlimited requested resources. This accelerates individual completions, reducing average job completion time (JCT). Furthermore, it preserves *fairness*: prioritized agents release resources earlier, and the fair-completion-order prevents starvation and head-of-line blocking, ensuring no agent finishes later than it would under strict fair sharing. Thus, selective pampering achieves both efficiency and fairness.

To validate this, we conduct a testbed experiment (Fig. 3) focusing on *KV cache* [27], a critical contended GPU memory resource. We submit two DocMerging agents simultaneously to a LLaMA2-7B model on a single A100 GPU. To intentionally induce contention, we restrict vLLM to 50% of total GPU memory, empirically profiled via Out-of-Memory (OOM) events to yield an allocation of 459 KV cache blocks.



(a) Instantaneous Fair Sharing (b) Pampering in Fair Order

Figure 3: KV block usage when running two DocMerging (DM) agents under different schemes.

As shown in Fig. 3a, under instantaneous fair sharing, each agent is restricted to use its fair share (as indicated by the KV block usage), resulting in an average *job (agent) completion time*, or JCT, of 210 s. However, if we prioritize the agents sequentially (based on their job completion order in Fig. 3a), as shown in Fig. 3b, the average JCT drops to 166 s—without delaying any single agent compared to the fair-sharing case in Fig. 3a. This verifies the effectiveness of *selective pampering*.

3 Solution

3.1 Overview

In this paper, we propose *Justitia*, a *fair* and *efficient* scheduler for task-parallel LLM agents using the *selective pampering* principle. However, applying this requires obtaining the agents' fair completion order (via service cost estimation) prior to execution, which brings multiple *challenges*. First, simply using compute amount (e.g., output length) fails to quantify the true service cost, as LLM inferences consume both compute and memory resources. Meanwhile, the auto-regressive nature of LLMs renders their resource demands highly uncertain a priori. Moreover, given the estimated costs, determining the ideal agent queuing order in an *efficient* (fast decision) yet *high-quality* (optimal performance) manner remains difficult.

Next in this section, we will elaborate how to address those challenges in *Justitia*. We first introduce our cost modeling method in Sec. 3.2, and then elaborate how to predict the agent-level resource demand in Sec. 3.3. Finally we describe the queuing strategy of *Justitia* in Sec. 3.4.

3.2 Memory-Centric Cost Modeling

The VTC work [9] adopts a compute-centric approach: it measures the serving cost as a weighted combination of the prefilling (input) tokens and decoding (output) tokens. However, such a method is over-simplified by ignoring the impact of KV cache consumptions. In fact, in modern LLM serving frameworks like vLLM [11], the GPU memory (*KV cache space*) is a dominant scheduling constraint for concurrent serving: new requests can be added to the *running* queue only when there is sufficient memory space; otherwise some requests would be interrupted and moved to the *swapped* queue. In that sense, inference serving cost shall be better depicted from the *memory* perspective.

To determine the agent scheduling order, we need to quantify the overall serving cost of each LLM agent, and the key is to describe the serving cost of an LLM inference. Moreover, the resource

demand of an LLM inference is not fixed during its lifetime. During the generative inference process, the sequence length keeps increasing, leading to increasing KV-cache occupations. Therefore, in this paper we propose a *memory-centric* cost modeling method that depicts the cumulative service cost of an LLM inference in both temporal and spatial dimensions.

To be specific, let p and d respectively represent the prefill and decode token length (Fig. 4), then we devise the cost metric, *KV token-time*, as:

$$c = \sum_{i=1}^{d}(p+i) = pd + \frac{d(d+1)}{2} \approx pd + \frac{d^2}{2}. \qquad (1)$$

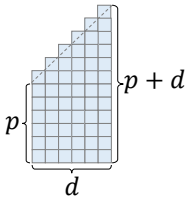


Figure 4: Demand modeling of an LLM inference. p and d are respectively the prefill and decoding token length. The service cost is then approximately depicted as $pd + d^2/2$.

This formula adds up the KV cache occupation[2] over all the iterations[3] of that inference; this is reasonable because each iteration is an independent service slot where different requests compete for the memory space. From the above formula, we can learn that there exists a *quadratic* relationship between the cost volume and the generation length (d), rendering the *linear* relationship [9] inaccurate.

Furthermore, the overall serving cost of an LLM agent can be naturally defined as the sum of the KV token-time of all its constituting inferences. Such a memory-centric modeling method can express the true serving cost of LLM agents.

3.3 MLP-based Demand Prediction

To support *selective pampering*, we must predict each agent's cost volume. Existing methods like S^3 [29] fine-tune language models (e.g., DistilBERT) for prediction. However, this is unsuitable for our context: fine-tuning a 66-million-parameter model demands scarce large datasets, running it incurs non-negligible runtime overhead, and a single model fails to capture the heterogeneous cost distributions across different agents.

Therefore, we design a *light-weight* and *sufficiently-accurate* cost prediction method. For *accuracy*, we exploit the empirical stability of agent resource demands (Appendix A) by maintaining a dedicated model for each agent type as *prior knowledge*. For *low overhead*, we utilize a Multi-Layer Perceptron (MLP) for each agent. Its simple structure enables efficient training with limited data and requires minimal computational resources for real-time scheduling.

Fig. 5 details this workflow. We first vectorize the input prompt using TF-IDF [30], a lightweight method extracting word importance with minimal overhead. This vector then feeds into the agent-specific 4-layer MLP to predict the cost. The first layer's neuron count is proportional to the average input size. Each model is efficiently trained on just 100 samples via gradient descent using Mean Squared Error with L2 regularization. Note that we do not assume identical demands across inputs of the same agent type: the prompt features capture input-dependent variation, and the predictor targets workload adaptation within each agent type rather than cross-agent generalization. Experimental results later in Sec. 4.4 demonstrate that this method achieves high accuracy with low overhead.

3.4 Fair and Efficient Agent Queuing

This section details *Justitia*'s queuing algorithm. To ensure inferences of high-priority agents are served consecutively without interleaving, each inference is queued based on its agent's overall demand. Furthermore, to avoid preemption overheads (e.g., KV cache swapping), we adopt vLLM's [11] non-preemptive scheduling principle[4]. Regardless of priority, pending requests cannot

[2]The unit is the number of KV cache blocks corresponded to one token (over all the LLM layers and heads). This is more *concise* for analysis than recording the exact KV block number, although both are essentially equivalent in scheduling.

[3]For simplicity we do not consider the time difference when generating different tokens. In realistic execution, an LLM sequence would be batched with sequences from other inference requests [28]; all such sequences jointly determine the per-iteration inference latency. The inference time of such runtime batches with mixed sequence is statistically stable.

[4]To prevent inference interference, vLLM is non-preemptive [11]: inferences halted by KV cache exhaustion enter a high-priority swapped queue and resume first. Thus, pending requests never preempt running inferences.

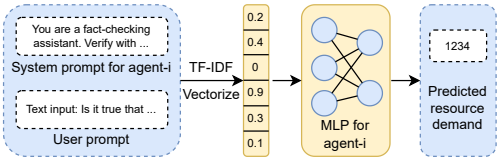


Figure 5: Service cost prediction for agents with an MLP model.

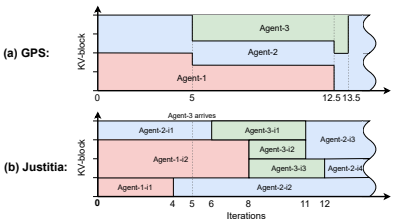


Figure 6: In *Justitia*, agents are prioritized according to their relative completion order under GPS.

preempt a running inference; agent-level preemption strictly occurs only after an entire inference completes. Moreover, pampering only changes the admission priority and does not reserve resources exclusively: when the prioritized agent cannot fully use the KV cache, requests from other agents are still admitted and batched together, keeping the system work-conserving. We next focus on how to obtain the agent queuing order in a practical manner.

GPS as the reference system. Recall that our insight is to conduct saturated agent serving following the fair completion order; the key is to acquire that fair completion order efficiently, at agent arrival time. One choice is to use the VTC scheduler [9] as the reference system. Yet, idealized fair sharing requires each agent to stick to its fair share, whereas an inference scheduled in VTC can take an arbitrarily large KV cache space as needed to accommodate its prompt, disrupting the expected execution status. In that sense, getting the VTC completion order requires a trial run (or simulation) based on the full agent demand knowledge (including those arriving later). Moreover, estimating the fair completion order requires continuously refreshing, upon the arrival or completion of any agent, the remaining resource demand and the latest resource fair share, which is cumbersome.

To overcome the cumbersome recalculation of fair completion orders, we directly adopt the classical *fair queuing* algorithm [15, 16]. In principle, fair queuing utilizes the completion order under an idealized fair sharing scheme, known as *Generalized Processor Sharing* (GPS), as the scheduling priorities. Under GPS, backend resources are arbitrarily divisible and equally allocated to each active tenant.

Virtual-time based calculation of fair queuing order. However, similar to the VTC approach, estimating the exact completion time under GPS at the moment of agent arrival is impractical without prior knowledge of runtime contention. To acquire the relative scheduling order in one shot at arrival time, fair queuing introduces the notion of *virtual time*. Instead of continuously updating each agent's expected fair completion time whenever the contention state changes, virtual time simply adjusts the rate at which system time elapses. We therefore adopt this virtual-time-based fair queuing method to schedule LLM agents efficiently.

Fig. 6 presents the key queuing mechanism of our proposed *Justitia* scheduler (for simplicity we ignore demand volume inflation during inference). For LLM agent scheduling, we also use the GPS scheme to gauge the fair completion order (i.e., the scheduling priority) of LLM agents. As shown in Fig. 6, suppose the agent completion order under the idealized fair-sharing system, GPS, is Agent-1, Agent-3, and Agent-2, then we set that order as their scheduling priority. In that sense, when Agent-2-i1 (the first inference request of Agent-2) completes, the requests from Agent-3 would take the service opportunity. Specifically, for efficient completion order estimation, we adopt the notion of virtual time $V(t)$, which is defined as a function of the real time t:

$$V(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}V(t) = M/N_t. \tag{2}$$

Here, M is the total KV cache space, and N_t is the number of active agents in GPS at time t; M/N_t thus represents the instantaneous fair share. Further, $V(t)$ increases at the marginal rate at which each agent receives service in GPS. When Agent-j arrives at time a_j (with C_j being its KV token-time cost), *Justitia* calculates its *virtual finish time* F_j, i.e., the virtual time at which the agent completes in GPS, as:

$$F_j = V(a_j) + C_j. \tag{3}$$

The virtual finish time of an agent, once calculated, requires no update in the future. While the arrival of later agents would change the fair-serving rate, they do not change the relative completion order

among existing agents (i.e., the relative order in $\{F_j\}$), because each active agent would always be serviced with the same rate. *Justitia* further adopts the $\{F_j\}$ order as the scheduling priority. In this way we can attain low scheduling overhead: the status refreshing overhead on agent arrival or completion is *constant*, and the complexity to select the next agent to serve is $O(\log N_t)$.

In other words, selective pampering trades instantaneous fairness for efficiency by serving agents in their GPS completion order, and the deviation from that fair reference remains bounded: we have mathematically proved that, under *Justitia* an agent can be guaranteed to complete within a constant delay after its completion under GPS (the virtual fair scheduler). Due to the space limitation, please refer to Appendix B for the detailed theorem and proof.

4 Evaluation

4.1 Setup

Hardware Platform. We have implemented *Justitia* atop vLLM [11], the mainstream LLM serving framework. Due to the space limitation, the implementation details are placed in Appendix C. In testbed experiments, we evaluate *Justitia* with diverse LLMs. The LLMs[5] we use are LLaMA-7B (served on an A100-PCIe-40GB GPU), LLaMA-13B [31] (served on four V100-PCIe-16GB GPUs in tensor-parallel mode), and Qwen2.5-32B [32] (served on an H800-PCIe-80GB GPU).

Workloads. For our experiment, we created a mixed workload suite with 300 LLM agents covering diverse task semantics (e.g., summarization, reasoning, and coding). For each agent type, we reuse the example prompt inputs from their original repositories. Due to space constraints, we defer the detailed descriptions and citations of all nine agent classes to Appendix D. Similar to prior work [33, 34], we categorize these agents based on execution time into small (usually less than 1 min), medium (between 1 and 10 mins), and large (longer than 10 mins) tasks. We set their sampling probabilities to 72%, 26%, and 2%, respectively. Regarding agent submission, we follow the request arrival time in the traces released by Mooncake [35], with the submission window respectively set to 6, 9, 18 mins (i.e., with workload intensity[6] respectively be $3\times$, $2\times$ and $1\times$).

Baselines. We compare *Justitia* with five baselines: (a) vLLM [11], which adopts FCFS policy at the inference level; (b) vLLM-SJF [14], which schedules LLM inferences following SJF policy with the DistilBERT-predicted inference durations; (c) Parrot [6], which adopts the FCFS policy at the agent level; (d) VTC [9], which seeks to approximate the instantaneous fair-sharing policy at the agent level; (e) SRJF, which uses our predicted serving cost to enforce the shortest-remaining-job-first policy at the *agent* level.

Metrics. *Justitia* is expected to behave well in both efficiency and fairness. Regarding efficiency, we adopt the average and P90 job completion time (JCT, meaning the duration between job arrival and completion); here a *job* refers to a running agent triggered by a given user input. Regarding fairness, we adopt the notion of *finish-time fair ratio*, which is the ratio between a job's completion time under a fair scheduler (we use VTC as the baseline) and its realistic completion time. Formally, for job j under scheduler $\mathcal{S}$, it is defined as $\rho_j^{\mathcal{S}} = T_j^{\mathrm{VTC}}/T_j^{\mathcal{S}}$, where $T_j^{\mathcal{S}}$ and T_j^{VTC} denote the JCT of job j under $\mathcal{S}$ and under VTC, respectively (i.e., the reciprocal of the finish-time fairness in [26], with VTC as the fair-sharing reference). $\rho_j^{\mathcal{S}} \geq 1$ means that job j completes no later than it would under instantaneous fair sharing; thus, the higher that ratio, the better the fairness level.

4.2 End-to-end Scheduling Performance

Efficiency performance. We first evaluate the efficiency performance of *Justitia*. Fig. 7 shows the overall serving performance with different models. Across all three backends, *Justitia* can substantially outperform the mainstream schedulers. Specifically, the average JCT under *Justitia* is 57.5%

[5]We employed diverse GPU configurations to demonstrate the generalizability of our approach.

[6]In our evaluation, we prioritize workload intensity over the absolute number of agents or physical cluster scale. The total agent count is merely a product of intensity and experimental duration. Crucially, workload intensity directly dictates queue size and generates the high-contention scenarios necessary to rigorously evaluate scheduler performance; simply extending duration does not increase scheduling difficulty. Furthermore, cluster scale effectively abstracts to workload intensity: higher concurrent submission rates in larger multi-GPU clusters directly manifest to the scheduler as heightened workload intensity.

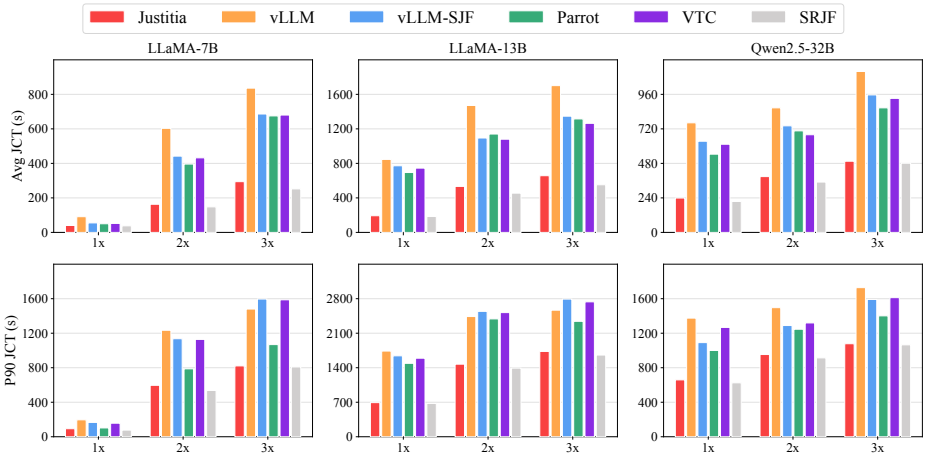


Figure 7: JCT performance with different LLM backends.

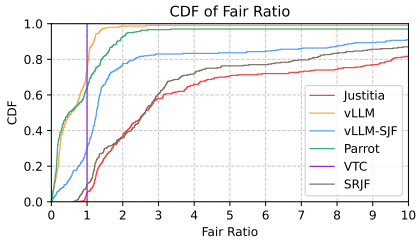


Figure 8: CDF of finish-time fair ratios (i.e., VTC-JCT normalized by realistic JCT; higher is better) under 3× intensity.



Figure 9: Behaviors when serving a large agent with many small ones.

(61.1%) better than that under VTC (Parrot). In the meantime, *Justitia* attains a very close JCT performance with SRJF, indicating that it can attain near-optimal efficiency.

Fairness performance. To evaluate fairness performance, Fig. 8 plots the CDF of all agents' finish-time fair ratios. It shows that 92% of agents can complete under *Justitia* no later than it would have under VTC (with a worst-case delay of only 26.0%, which is much smaller than others). This confirms the fairness superiority of *Justitia*. Interestingly, we find that SRJF can also attain a relatively good fairness performance, because it can avoid head-of-line blocking, and many prioritized agents cannot saturate the service backend, so the remaining KV resources are multiplexed by others, exhibiting a modest fair-sharing effect. Yet, SRJF may starve the large agents, which we next verify with a micro-benchmark experiment.

4.3 Micro-benchmark Experiments

Starvation avoidance under *Justitia*. A well-known deficiency of SRJF is the starvation problem: when short agents keep arriving, the execution of long agents may be infinitely delayed. To confirm that, we conduct a micro-benchmark experiment. Specifically, we first submit a large "elephant" agent, followed by a continuous stream of lightweight "mice" agents issued at one-second intervals. Detailed configurations and the specific task types for these agents are provided in Appendix D. In Fig. 9, we show the relationship between the JCT of the elephant agent and the number of "mice" agents respectively under SRJF and *Justitia*. It clearly demonstrates that, with an increasing number of mice agents, the elephant agent may potentially be delayed forever under SRJF; in contrast, that delay under *Justitia* is bounded regardless of the number of competing agents.

Performance robustness against prediction errors. In reality, it is still possible that the predicted cost of an LLM agent is highly inaccurate. To check the performance robustness of *Justitia* against prediction errors, we resort to another experiment with controlled prediction errors. Given the profiled cost ground-truth (*temperature* set to 0 to ensure recurrence), we scale it with random factors

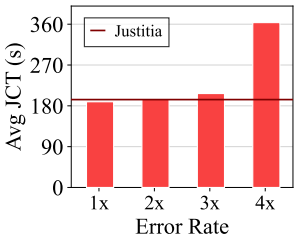


Figure 10: *Justitia* performance when prediction error amplifies.

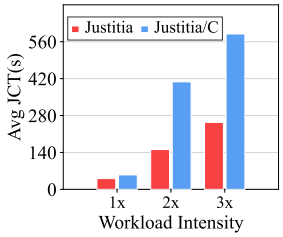


Figure 11: Performance comparison between different cost modeling methods.

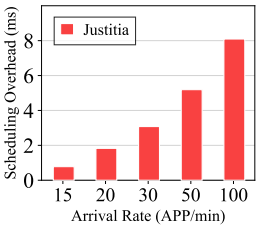


Figure 12: *Justitia* scheduling delay at different request arrival rates.

Table 1: Performance comparison between MLP and DistilBERT in prediction under 2× workload intensity. MLP and DistilBERT model training is conducted on a V100 GPU.

Prediction Model	**Avg. Relative Error (%)**	**Avg. Inference Overhead(ms)**	**Avg. JCT (s)**	**Training Time**
MLP	53.0	2.16	151.1	∼1 min
DistilBERT	452	55.7	366.7	∼2 h

at different levels. As in Fig. 10[7], $\lambda\times$ means that the original cost information is scaled by a random factor in $[1/\lambda, \lambda]$ before being used by *Justitia* (when λ is 1 we directly use the ground-truth). Fig. 10 shows that the performance of *Justitia* is highly robust: compared with the ideal case ($\lambda = 1$), the average JCT is inflated by only 9.5% when λ is set to 3. To provide a more comprehensive evaluation, we have extended our robustness analysis to encompass both fairness metrics and additional baselines that rely on cost prediction. Detailed figures are provided in Appendix E.

4.4 Ablation Study

Necessity of memory-centric cost modeling. To check the necessity of memory-centric cost modeling, we introduce a variant of *Justitia* by replacing its cost model with that of VTC (i.e., $p+2d$ [9]), which we call *Justitia*/C. We repeat the experiment in Fig. 7 (LLaMA-7B), and compare the average and P90 JCT under *Justitia*/C with the vanilla *Justitia*. As shown in Fig. 11, compute-centric cost modeling would incur a JCT performance degradation of up to 42.3%, which confirms the necessity to adopt memory-centric cost modeling.

Effectiveness of MLP-based demand prediction method. To check the superiority of MLP-based demand prediction, we introduce another *Justitia* variant: *Justitia*-S3, which uses a DistilBERT model for cost prediction in *Justitia*. In *Justitia*-S3, the DistilBERT model is trained on the same dataset as the MLP-based models. In Table 1 we compare the two methods in multiple aspects: average prediction error (prediction gap normalized by the ground-truth), average prediction overhead, average JCT, as well as the model training time. Table 1 suggests that our MLP-based method can remarkably outperform the DistilBERT-based method.

4.5 Overhead Analysis

The overhead in *Justitia* stems from two sources: (1) the one-shot prediction performed when an agent arrives, and (2) the updating of system virtual time and the selection of highest-priority agent upon the arrival or completion of any agent. As can be seen also in Table 1, the prediction overhead for a single new agent is approximately 2.16 ms, which is a negligible cost. Fig. 12[8] further demonstrates the average scheduling latency of *Justitia* under varying arrival rates (reflecting different scheduling scales). The results confirm that the scheduling overhead remains consistently low (under 10 ms) in all scenarios. Since agent JCTs are at the scale of tens to hundreds of seconds (Fig. 7), such millisecond-level overhead has a negligible impact on end-to-end JCT even under high arrival rates.

[7]The sudden jump at 4× error rate relates to the workload composition: we find MRS and ALFWI are two influential agent types in our benchmark, which approximately have a 10× cost difference. At 4× error rate, their relative execution orders begin to be reversed, which disproportionately inflates the average JCT due to their cost disparity.

[8]While the time to compute MLP prediction and to update the virtual counter is constant, the time to find the *best* scheduling candidate is not, which increases monotonically with the queue length and cannot be avoided by any efficiency-centric schedulers.

5 Conclusion

In this paper, we propose *Justitia*, an efficient and also fair scheduler for task-parallel LLM agents. *Justitia* works by scheduling LLM agents in a selective pampering manner, following their fair completion order. Specifically, it quantifies the true service cost of LLM agents in a memory-centric manner, and adopts an MLP-based method to predict the agent service cost a priori. Moreover, *Justitia* adopts the fair queuing method to efficiently get the expected agent completion order under an idealized fair scheduler. Testbed measurements with diverse workloads confirm that *Justitia* can behave well in both fairness and efficiency.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China (NSFC), under Grant No. 62572304. The authors declare no competing interests.

References

[1] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

[2] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

[3] Qiuhan Gu. Llm-based code generation method for golang compiler testing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 2201–2203, 2023.

[4] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

[5] Kimi. Kimi agent swarm. `https://www.kimi.com/blog/agent-swarm`, 2026. Accessed: 2026-05-06.

[6] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, 2024.

[7] Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Aurick Qiao, and Hao Zhang. Efficiently serving llm reasoning programs with certaindex. *arXiv preprint arXiv:2412.20993*, 2024.

[8] Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of llm-based applications with ayo. In *ACM ASPLOS*, 2025.

[9] 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 (OSDI 24)*, 2024.

[10] Redwan Ibne Seraj Khan, Kunal Jain, Haiying Shen, Ankur Mallick, Anjaly Parayil, Anoop Kulkarni, Steve Kofsky, Pankhuri Choudhary, Renèe St. Amant, Rujia Wang, Yue Cheng, Ali R. Butt, Victor Rühle, Chetan Bansal, and Saravan Rajmohan. Ensuring Fair LLM Serving Amid Diverse Applications. *arXiv e-prints*, page arXiv:2411.15997, November 2024.

[11] 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.

[12] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

[13] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. Efficient interactive llm serving with proxy model-based sequence length prediction. *arXiv preprint arXiv:2404.08509*, 2024.

[14] Rana Shahout, Eran Malach, Chunwei Liu, Weifan Jiang, Minlan Yu, and Michael Mitzenmacher. Don't stop me now: Embedding based scheduling for llms. In *ICLR*, 2025.

[15] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4):1–12, 1989.

[16] Abhay K Parekh and Robert G Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM transactions on networking*, 1(3):344–357, 1993.

[17] Junchao Wu, Shu Yang, Runzhe Zhan, Yulin Yuan, Lidia Sam Chao, and Derek Fai Wong. A survey on llm-generated text detection: Necessity, methods, and future directions. *Computational Linguistics*, pages 1–66, 2025.

[18] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. Smart-llm: Smart multi-agent robot task planning using large language models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12140–12147. IEEE, 2024.

[19] Gemini: Next-generation AI model. `https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#sundar-note`, 2024.

[20] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

[21] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[22] I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, et al. Factool: Factuality detection in generative ai–a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*, 2023.

[23] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[24] 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.

[25] Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient llm scheduling by learning to rank. *arXiv preprint arXiv:2408.15792*, 2024.

[26] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient {GPU} cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 289–304, 2020.

[27] Shiwei Gao, Youmin Chen, and Jiwu Shu. Fast state restoration in llm serving with hcache. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 128–143, 2025.

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

[29] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S^3: Increasing GPU Utilization during Generative Inference for Higher Throughput. *arXiv e-prints*, page arXiv:2306.06000, June 2023.

[30] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[31] 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.

[32] Qwen. Qwen2.5-32B. `https://huggingface.co/Qwen/Qwen2.5-32B`, 2024. Accessed: 2026-01-29.

[33] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

[34] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *ACM SOSP*, 2023.

[35] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.

[36] Langchain. `https://github.com/langchain-ai/langchain`, 2025.

[37] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *NeurIPS*, 2023.

[38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[39] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *NeurIPS*, 2022.

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

[41] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

[42] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *USENIX OSDI*, 2024.

[43] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[44] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[45] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

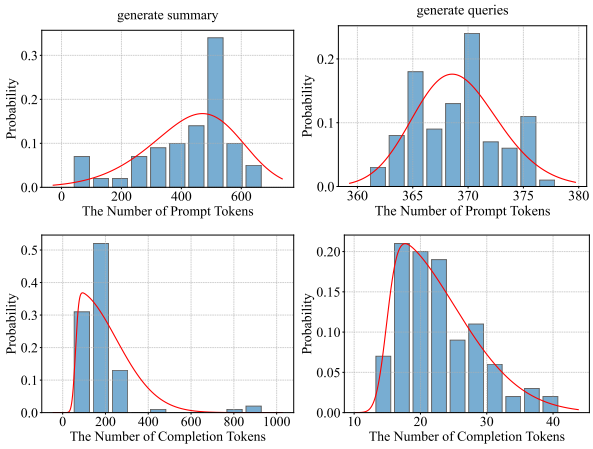
[46] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[47] Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. Infercept: Efficient intercept support for augmented large language model inference. In *Forty-first International Conference on Machine Learning*, 2024.

[48] Yi Zhai, Dian Shen, Junzhou Luo, and Bin Yang. Toolcaching: Towards efficient caching for llm tool-calling. *arXiv preprint arXiv:2601.15335*, 2026.

A Agent-specific Demand Stability

In this section, we elaborate that the resource demands of an LLM agent are relatively stable across different trial runs (each with a distinct agent input). In fact, since each LLM agent is designed for specific use cases, the resource demands of which are relatively stable across different execution runs. Fig. 13 shows the execution information (input/output length) of specific inference stages in the two LLM agents over 100 trial runs, which exhibits strong demand correlations. For example, `generate-queries` inferences in the Knowledge-Based-Query-Answering Verification (KBQAV) agent all have an input token length between 360 and 380.



(a) Summarization (b) KBQA Verification

Figure 13: Prompt and decoding length distribution respectively for the `generate-summary` and `generate-queries` inferences in the MapReduce-Summarization (MRS) agent and the Knowledge-Based-Query-Answering Verification (KBQAV) agent. In each case, we divide the length range into 10 buckets, and calculate the value appearance probability in each bucket (accompanied by the fitted curves assuming skewed Gaussian distribution).

B Delay Analysis

In this part we analyze the fairness properties of *Justitia*. In practice, an inference cannot be started if the available KV block volume is smaller than its prompt size, which may cause the memory fragmentation problem. For ease of analysis, we assume the prompt size of typical inferences is much smaller in scale than the total KV cache space. Our measurement shows that, in the released Mooncake dataset [35], each request in average only takes 3.20% of the total KV cache size when running LLaMA-3.3 70B on H100 (with 28.65GB KV cache space). Therefore, we can faithfully neglect the fragmentation problem. That is, if there are inferences waiting in the queue, all the KV-blocks on the server would be fully utilized.

Symbols. We let agents be indexed in ascending order of their inference start time. That is, app-j is the j^{th} agent that is allocated KV-blocks in *Justitia*. Meanwhile, we let C_j represent the total KV token-time of app-j, which is the sum of all the KV token-time of all its inferences. We also let $C_{\max}$ denote the maximum total KV token-time among all agents. Similarly, we let $c_{\max}$ represent the maximum KV token-time required by any single inference across all agents. Finally, we let f_j denote the completion time of app-j in *Justitia*, and $\bar{f}_j$ the GPS completion time. Table 2 summarizes the notations used in the analysis. Given these definitions, we then establish the constant delay bound of *Justitia* through the following theorem.

Theorem B.1 (Constant delay bound)**.** *With Justitia, an agent is guaranteed to complete within a constant time after its completion in GPS, i.e., for each app-j, we have*

$$f_j - \bar{f}_j \leq 2c_{\max} + C_{\max}/M. \tag{4}$$

Proof. Our analysis critically focuses on the *slowdown period* of an agent. In particular, we say an agent is *slowed down* at time t if it has a *backlogged* inference waiting for service. In other words, at

Table 2: Summary of important notations and definitions.

M	The number of KV-blocks in a server
$\bar{f}_j$	The agent completion time for app-j in GPS
f_j	The agent completion time for app-j in *Justitia*
C_j	The total KV token-time required by app-j
$C_{\max}$	The maximum KV token-time required by any agent
$c_{\max}$	The maximum KV token-time required by any single inference
a_j	The arrival time of app-j
e_j	The ending time of the slowdown period of app-j

any moment in the slowdown period, the agent could have run more inferences if receiving more KV-blocks. Because slowdown delays the agent completion, bounding the timespan of the slowdown periods is the key to analyzing the longest possible delay.

For each app-j, we let a_j be the arrival time of app-j. Depending on the number of available KV-blocks at time a_j, app-j is either slowed down, or allocated a sufficient number of KV-blocks to run all inferences right after the arrival. In particular, we let e_j be the time when the slowdown period of app-j ends. We then have $a_j < e_j$ if app-j experiences slowdown, and $a_j = e_j$ otherwise. In either case, after the slowdown period, app-j runs all the backlogged inferences in parallel, and is guaranteed to complete after at most the maximal inference runtime, i.e.,

$$f_j \leq e_j + c_{\max}. \tag{5}$$

Therefore, to bound the agent completion time, it is critical to analyze when the slowdown period ends (i.e., e_j).

We consider the most general case where app-j is slowed down right after the arrival, i.e., $a_j < e_j$. During the slowdown period $[a_j, e_j]$, all the KV-blocks are busy. While *Justitia* preferentially offers KV-blocks to agents in ascending order of their GPS completion times, agents may start services out of order due to the dynamic arrivals. In particular, an agent that completes before app-j in GPS may arrive late, after app-j starts in *Justitia*. Let $\mathcal{A}$ be the set of all these agents, i.e., $\mathcal{A} = \{k \mid k > j \text{ and } \bar{f}_k \leq \bar{f}_j\}$.

On the other hand, an agent that completes after app-j in GPS may start earlier in *Justitia*, before app-j arrives. Let app-m be such an agent that is serviced *the most recently*. That is, m is the *largest* integer satisfying both $m \leq j-1$ and $\bar{f}_m > \bar{f}_j$, i.e., $\bar{f}_m > \bar{f}_j \geq \bar{f}_k$ for all $m < k < j$. In other words, app-m completes after agents $m+1, \ldots, j$ in GPS, but is allocated KV-blocks before all these agents in *Justitia*. We let $\mathcal{B}$ be the set of all these agents that complete no later than app-j (hence earlier than app-m) in GPS but start after app-m in *Justitia*, i.e., $\mathcal{B} = \{k \mid m < k \leq j\}$.

By definition, agents in $\mathcal{A}$ and $\mathcal{B}$, though completing earlier than app-m in GPS, are serviced no earlier than app-m in *Justitia*. These agents must have not yet arrived before app-m is allocated KV-blocks—otherwise *Justitia* would have serviced them before app-m. We then have

$$\min_{k \in \mathcal{A} \cup \mathcal{B}} \{a_k\} \geq b_m, \tag{6}$$

where b_m is the first time when app-m is allocated KV-blocks in *Justitia*. This suggests that since b_m, GPS has completely serviced, *at least*, all agents in $\mathcal{A} \cup \mathcal{B}$ by the time app-j finishes in GPS, i.e.,

$$\bar{f}_j \geq b_m + \tfrac{1}{M} \textstyle\sum_{k \in \mathcal{A} \cup \mathcal{B}} C_k. \tag{7}$$

We next analyze e_j, the ending time of the slowdown period of app-j in *Justitia*. During time interval $[b_m, e_j]$, *Justitia* may have completed all the agents in $\mathcal{A}$ and $\mathcal{B}$, along with at most $M-1$ inferences requiring a maximal runtime. In addition, app-m is allocated KV-blocks at b_m, and may also complete before e_j. Finishing these works using all M KV-blocks takes at most

$$e_j \leq b_m + \tfrac{1}{M}[\textstyle\sum_{k \in \mathcal{A} \cup \mathcal{B}} C_k + C_m + (M-1)c_{\max}]. \tag{8}$$

Plugging (8) into (5) and subtracting (7), we have

$$f_j - \bar{f}_j \leq c_{\max} + \tfrac{M-1}{M} c_{\max} + C_m/M < 2c_{\max} + C_{\max}/M.$$

This completes the proof. □

C Implementation Details

We have implemented *Justitia* atop vLLM [11], the mainstream LLM serving framework. The overall workflow with *Justitia* is shown in Fig. 14.

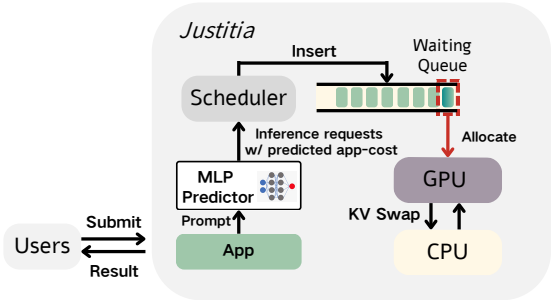


Figure 14: *Justitia* Workflow

As elaborated in Fig. 14, each time a user submits an application request, the application-specific (MLP) predictor would first predict the application-level service cost of that application based on user input in the ongoing run (following the method described in Sec. 3.3). The *Justitia* scheduler would then calculate the virtual completion time of that application—based on its predicted service cost and the current virtual time (as described in Sec. 3.4). That application-level virtual completion time would be used as the scheduling priority of its inference tasks when inserting them into the task waiting queue. For each scheduled inference task, we serve it following the standard vLLM logic. Each time the KV cache space is full, *Justitia* moves *some* tasks to the swapped queue, with their KV contents temporarily swapped to CPU memory; meanwhile, no new tasks can be admitted to the running queue when there are pending tasks in the swapped queue.

D Workload Setup

We evaluate our system using a diverse set of agent classes: (a) MapReduce Summarization (MRS) [36], (b) Plan-and-Execution (PE) [37], (c) Code Checking (CC) [22], (d) Knowledge-Based-Query-Answering Verification (KBQAV) [22], (e) Equation Verification (EV) [22], (f) Fact Verification (FV) [38], (g) ALFWorld Interaction (ALFWI) [38], (h) Document Merging (DM) [20], and (i) Self Consistency (SC) [23].

Following prior workload characterizations [33, 34], we categorize these agents into three classes based on their execution times: *small* (EV, FV, CC, ALFWI, and KBQAV; < 1 minute), *medium* (PE and SC; $1 - 10$ minutes), and *large* (DM and MRS; > 10 minutes). To reflect realistic distributions, we set their sampling probabilities to 72%, 26%, and 2%, respectively.

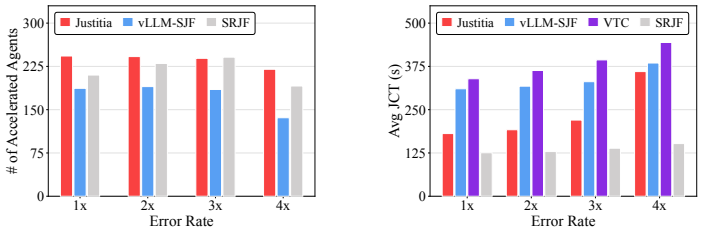
For agent submissions, we simulate realistic request arrival times using production traces from Mooncake [35]. To evaluate workload intensities of $3\times$, $2\times$, and $1\times$, we set the corresponding submission windows to 6, 9, and 18 minutes, respectively.

Finally, in our micro-benchmark experiments, we utilize representative LLM agent tasks to simulate distinct workload dynamics. Specifically, we employ an MRS task to represent "elephant" agents, which process long documents and generate heavy, sustained computational demand. Conversely, to represent diverse, short-lived user requests ("mice"), we randomly sample tasks from the KBQAV, CC, and ALFWI categories.

E Additional Experimental Results

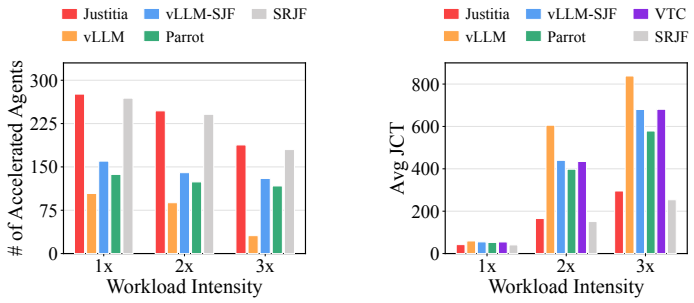
E.1 Robustness Experiments for the Fairness Metric and Other Baselines

To provide a more comprehensive evaluation, we extend our robustness analysis to encompass fairness metrics and additional cost-prediction baselines. Specifically, we quantify fairness by measuring the proportion of tasks that achieve earlier completion times compared to the VTC scheduler



(a) Fairness performance (b) Efficiency performance

Figure 15: Fairness and efficiency performance as prediction error amplifies



(a) Fairness performance (b) Efficiency performance

Figure 16: Fairness and efficiency performance with LLaMA-7B on V100.

across varying prediction error rates. As shown in Figure 15a, *Justitia* consistently achieves superior fairness over all baselines, even when subjected to injected prediction errors.

Furthermore, we evaluate the robustness of other prediction-dependent baselines using the experimental configuration established for Figure 10. As demonstrated in Figure 15b, while all evaluated baselines experience a certain degree of efficiency degradation as the error rate increases, *Justitia* maintains significantly higher overall performance and superior fairness.

E.2 Hardware Portability Analysis

To demonstrate the hardware-agnostic nature of our solution, we replicate the end-to-end evaluation (originally shown in Figure 7) using the LLaMA-7B model deployed on NVIDIA V100 GPUs, rather than A100s. The resulting efficiency and fairness metrics, detailed in Figures 16b and 16a, align perfectly with the performance trends observed on the A100 platform.

E.3 P90 JCT Results

In this subsection, we present the 90th percentile Job Completion Time (P90 JCT) to demonstrate the system's tail performance.

The following two Figures 17 and 18 supplement Figure 10 and 11 in the main text, which only report avg JCT, by additionally presenting the P90 JCT results. The conclusions are consistent with those observed from the avg JCT results.

F Additional Related Works

Apart from the scheduling works in Sec. 2.2, many additional works seek to accelerate individual LLM requests in a series of aspects. For example, FlashAttention [39] and FlashDecoding [40] algorithms seek to accelerate LLM inference at the operator level; model-quantization [41] and prefill-decode-separation [42] methods have been proposed to maximize backend utilization; Speculative decoding methods like [43] are also applied to improve the inference throughput by accelerating the decoding process.

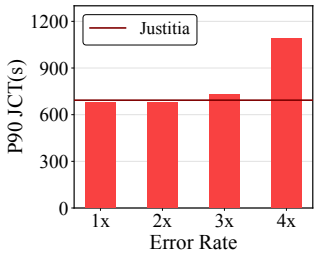


Figure 17: p90 result for Figure 10

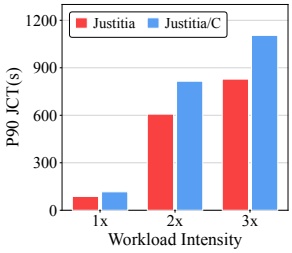


Figure 18: p90 result for Figure 11

Meanwhile, agent serving is also a popular research topic in the recent literature. Apart from the task-parallel nature which is the focus of this paper, modern agents may also stand out by their tool usage capabilities, as exemplified by *Toolformer* [44] (API calling for arithmetic/fact-checking), *Gorilla* [45] (large-scale API libraries), and *ToolLLM* [46] (multi-tool reasoning frameworks). For those tool-usage agents, some other optimization techniques have also been proposed [47, 48]. It can be expected that both task-parallel nature and tool-diversifying nature would be common in future agent design, thus their optimization techniques can be jointly applied.

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 summarize the paper's scope and contributions: Justitia targets fair and efficient scheduling for task-parallel LLM agents using selective pampering, memory-centric cost modeling, MLP-based demand prediction, and virtual-time-based fair queuing. The main claims are supported by the design in Section 3, the theoretical delay analysis in Appendix B, and the experimental results in Section 4.

   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: The paper discusses key limitations and boundary conditions through its assumptions and robustness analysis. For example, Appendix B states the assumption that prompt sizes are small relative to the total KV cache so that fragmentation can be neglected.

   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: The main theoretical result is stated as Theorem B.1, which establishes a constant delay bound for Justitia relative to the GPS fair scheduler. Appendix B provides the relevant assumptions, notation, and complete proof for this result.

   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: The paper discloses the main information needed to reproduce the experimental results, including the evaluated LLM backends and GPU platforms, workload composition, arrival traces, baselines, and evaluation metrics in Section 4.1. Appendix C further describes the implementation workflow of Justitia atop vLLM.

   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: The submission includes anonymized code and reproduction scripts in the supplemental material for the main experimental results.

   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: The paper specifies the main experimental settings, including the evaluated LLM backends, GPU platforms, workload composition, request arrival traces, baselines, and metrics in Section 4.1. The training details for the MLP-based demand predictor, including the input representation, model structure, training data size, loss, optimizer, and regularization, are described in Section 3.3.

   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: [No]

Justification: The paper reports aggregate metrics such as average JCT, P90 JCT, finish-time fair ratio, and scheduling overhead, but does not include error bars, confidence intervals, or statistical significance tests. The experiments are conducted on controlled workload traces and system configurations, and the main conclusions are based on consistent performance trends across models, workload intensities, and ablation studies.

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: The paper reports the main compute resources used in the experiments, including the LLM backends and GPU platforms: LLaMA-7B on an A100-PCIe-40GB GPU, LLaMA-13B on four V100-PCIe-16GB GPUs, and Qwen2.5-32B on an H800-PCIe-80GB GPU, as described in Section 4.1. The paper also reports runtime overheads of Justitia, including prediction overhead and scheduling latency, in Section 4.5.

   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: [No]

    Justification: The paper does not include a dedicated discussion of broader societal impacts. Its primary focus is a systems scheduling problem for improving efficiency and fairness in shared LLM serving.

    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: The paper does not release a high-risk model or dataset that requires special safeguards for responsible release.

    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: [No]

Justification: The paper cites the existing systems, models, workloads, and traces used in the evaluation. However, it does not explicitly list the licenses and terms of use for all existing assets.

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 submission includes new code and reproduction scripts for Justitia in the supplemental material.

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 experiments or 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 or research with human subjects, so IRB approval or equivalent review is not applicable.

    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 of the paper is a scheduling algorithm for task-parallel LLM agents and does not use LLMs as an important, original, or non-standard component of the research methodology. LLMs are used as evaluated serving backends and workloads rather than as part of the method development.

    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.