SageSched: Efficient LLM Scheduling Confronting Demand Uncertainty and Hybridity

Zhenghao Gan Yichen Bao Yifei Liu Chen Chen* Quan Chen Minyi Guo

Shanghai Jiao Tong University
{ganzhenghao,yc.bao,nephrencake,chen-chen}@sjtu.edu.cn
{chen-quan,guo-my}@cs.sjtu.edu.cn

Abstract

Efficient LLM inference scheduling is crucial for user experience. However, LLM inferences exhibit remarkable demand *uncertainty* (with unknown output length beforehand) and *hybridity* (being both compute and memory intensive). Existing LLM schedulers rely on simple heuristics or focus purely on compute resources, thus suffering from suboptimal performance.

In this work, we propose *SageSched*, an efficient LLM scheduler that properly handles demand uncertainty and hybridity of inference workloads. *SageSched* combines prompt contents with the past inference results to predict the output-length distribution in a light-weight and also accurate manner. Meanwhile, it models the true service cost of an inference request with both compute and memory aspects considered. Finally, *SageSched* employs an uncertainty-aware scheduling policy that can yield the best overall efficiency given the request cost distributions. Testbed experiments over diverse setups confirm that *SageSched* can attain an efficiency improvement of over 28.7%.

1 Introduction

Large language models (LLMs) have revolutionized diverse domains, from conversational assistants [35, 9] to agentic and embodied systems [5, 46]. With the widespread adoption of LLMs, massive LLM inference requests are often concurrently served on GPU servers [43]. When scheduling LLM requests, it is increasingly crucial to minimize their overall service latency, or *Time-to-Last-Token* (TTLT), to optimize the end-to-end user experience [41, 38].

Nonetheless, contrary to conventional workloads, LLM inferences have two challenging characteristics in resource demands: *uncertainty* and *hybridity*. On the one hand, due to the auto-regressive nature, the output token length of an LLM inference is *non-deterministic* beforehand, exhibiting strong uncertainty [51]; in contrast, OS [30] or big data [36] workloads are often recurring with stable resource demands. On the other hand, with vast matrix-based operations and heavy KVCache reliance [37], LLM inferences are both *compute* and *memory* intensive; in contrast, with virtual memory techniques [1], OS schedulers only care about compute cost in process scheduling.

Confronting the demand uncertainty and hybridity of LLM inference workloads, existing LLM schedulers fail to attain high efficiency. Mainstream LLM frameworks like vLLM [25] and SGLang [53] adopt the classical *first-come-first-serve* policy, which often inflates the overall TTLT due to *head-of-line blocking*. While some recent schedulers [41, 38, 13] seek to approximate the *shortest-job-first* policy with the predicted output length, they are still suboptimal. First, those schedulers commonly rely on a fine-tuned model to directly predict the inference output length, which is both *heavy-weight* (due to costly training and predicting) and *inaccurate* (due to the difficulty of emulating the original

*Corresponding author.

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

generation effect). Meanwhile, existing schedulers queue the pending requests based on their output token length, which ignores demand hybridity and fails to capture their true service costs. Moreover, those schedulers commonly use a single (expected mean) value as the queuing index, failing to leverage the inherent uncertainty distribution information for the best queuing effect.

In this paper, we design *SageSched*, an LLM scheduler that attains high efficiency by properly handling the demand uncertainty and hybridity characteristics of inference workloads. There are three key techniques in *SageSched*. *First*, in demand prediction, motivated by the correlation between prompt similarity and output-length similarity, we propose *semantic-aware history-based predictor*, which combines the *prompt contents* with the *past inference results* to predict an output-length distribution. In this way, we can avoid the burden of maintaining a heavy-weight prediction model for each LLM or the complexity of directly emulating the original generation effect. *Second*, in cost modeling, we consider resource contention in both *compute* and *memory* aspects. By respectively studying the LLM serving process in compute- and memory-bound scenarios, we build a *unified cost model* that can always capture the true service cost of an LLM inference. *Third*, given that the service cost of an inference is best depicted as a distribution, we adopt an *uncertainty-aware request scheduling policy*. Specifically, when queuing each request, we calculate the *Gittins index* out of its cost distribution, which is known to yield the best overall performance for *jobs with unknown durations but known duration-distributions* [16]; we also periodically refresh ongoing inferences' Gittins indices to ensure timeliness of the scheduling plan.

We have implemented *SageSched* on top of the popular vLLM framework [44], and further evaluated its performance with diverse real-world LLM traces. Our experiments show that *SageSched* surpasses state-of-the-art LLM schedulers by over 28.7% in terms of the average TTLT. Besides, our deep-dive analysis further confirms the superiority of each technique (prediction, modeling, and scheduling) adopted in *SageSched*. Moreover, large-scale simulations confirm that *SageSched* can scale well in large clusters with relatively low scheduling overheads.

2 Background and Motivation

2.1 The LLM Scheduling Problem

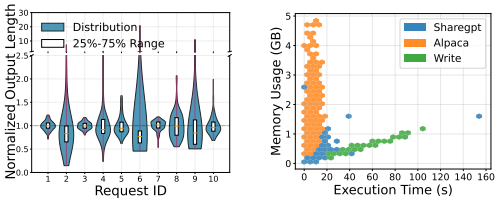
Demand *uncertainty* and *hybridity* of LLM inference workloads. Large language models (LLMs) [9, 33] have nowadays been widely adopted in diverse scenarios like chatbot conversation [35], code generation [6] and embodied intelligence [45]. Notably, LLM inferences are conducted in an auto-regressive manner [51]. As shown in Fig. 1(a), even a fixed prompt would generate different[2] output tokens in different runs. In that sense, the ultimate output length of an inference request can only be known after inference completion, exhibiting strong *uncertainty*.

Apart from demand uncertainty, demand *hybridity* is also a distinct property of LLM inference workloads. Specifically, during the inference process, each inference commonly caches its token sequence in GPU memory (which is called KVCache [37]) to avoid the recomputation overheads; provisioning such a memory space is a prerequisite for inferences to proceed in mainstream LLM frameworks like vLLM [44]. In Fig. 1(b), we profile the memory usage and execution time of all the requests respectively from three typical LLM datasets (*as selected* by [19])—*ShareGPT* [22], *Alpaca-Summarization* [21], and *Document-Write* [23]. It shows that both *computation* and *memory* resources are first-class citizens in LLM serving, unlike conventional compute-dominated OS workloads [14].

Scheduling is crucial on LLM service backends. LLM inference requests from diverse users are often served in a shared backend, where different requests compete for the limited compute and memory resources [40]. For good user experience, it is crucial to properly schedule the LLM inferences to achieve low overall latency. Specifically, as in a series of existing works [41, 38], given the increasing popularity of agentic LLM workloads, in this paper we primarily focus on optimizing the time-to-last-token (TTLT) metric, which is more significant[3] for non-human users than conventional metrics like time-to-first-token (TTFT) or time-per-output-token (TPOT).

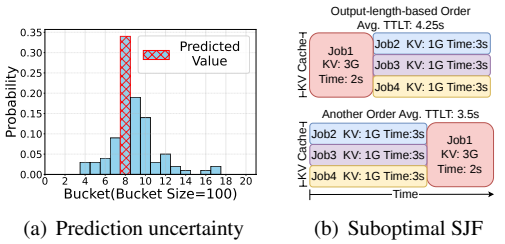
[2] In this paper, we use a temperature of 0.6 in all the inferences, a default setup for many popular LLMs [3, 39]. Note that the effectiveness of our solution does not rely on the temperature setup: even at $T = 0$, where the variance bar vanishes, the output length must still be predicted, since we do not assume exactly-recurring prompts.

[3] For example, for LLM-powered robotic control [5, 46], downstream operations can only be launched after generation completion. Meanwhile, optimizing TTLT often also helps to reduce the other metrics. For TTFT, by avoiding head-of-line



(a) Output uncertainty (b) Demand hybridity

Figure 1: Empirical evidence on the *uncertainty* and *hybridity* characteristics of LLM inference resource demands.



(a) Prediction uncertainty (b) Suboptimal SJF

Figure 2: Examples elaborating the deficiencies of existing schedulers when confronting demand uncertainty and hybridity.

2.2 Limitations of Existing LLM Schedulers

In the literature, a series of schedulers have been proposed for LLM inference workloads (due to the space limitation, we place the detailed elaboration of more related works in Appendix A.3). Confronting demand uncertainty, early-stage LLM schedulers usually employ *demand-agnostic* scheduling heuristics. Production LLM frameworks like vLLM [25] and SGLang [53] employ a simple *first-come-first-serve* (FCFS) scheduler, which suffers from head-of-line blocking. To address that problem, FastServe [48] adopts a *multi-level feedback queue* (MLFQ) to periodically degrade ongoing inferences to lower priorities after fixed service quanta—essentially approximating (yet not as good as) *shortest-remaining-processing-time* (SRPT) at the cost of additional preemption overheads.

Noticing the deficiencies of *demand-agnostic* schedulers, recent LLM schedulers adopt a *prediction-based* methodology. For example, the *Speculative Shortest Job First* (SSJF) [38] scheduler approximates *shortest-job-first* (SJF) by using a fine-tuned BERT-base model, which predicts the output token length given the inference prompt. In the meantime, Fu *et al.* [13] also approximate SJF using an OPT-125M model, which predicts the requests' relative output-length *rank* (rather than the exact lengths). Recently, TRAIL [41] approximates SRPT with an MLP model, which predicts the output length continuously at iteration granularity—using the feature embeddings of certain LLM layers.

However, the above prediction-based schedulers still fall short in handling demand *uncertainty*. First, their length predictors require heavy-weight offline training (fine-tuning typically requires collecting and processing 10k+ training examples [24, 13]), and the model trained for one LLM cannot be directly applied to another. Second, those output-length predictors yield a *single* prediction value, losing valuable probabilistic distribution information. In fact, demand uncertainty is a built-in property of LLM inferences, and a single prediction value would hardly be accurate in reality. As shown in Fig. 2(a), the single value predicted by DistilBERT as in [38] only achieves an accuracy of 34.1% (by hitting only one out of 20 buckets in total). As we show later in Sec. 3.3, preserving the length distribution can help to make better scheduling decisions. Therefore, in handling demand uncertainty, we need to design a *training-free* method that directly predicts the output-length *distribution*.

Meanwhile, we note that those schedulers also fall short in handling demand *hybridity*. They quantify an LLM request's service cost merely as its computation time (represented by its output token length), being inaccurate by neglecting its resource consumption of GPU memory. In fact, since both the input and output tokens take up KVCache space during inference, two inferences with the same output length may exhibit substantially different memory costs. As shown in Fig. 2(b), when the service backend is bottlenecked on GPU memory, simply prioritizing the LLM inferences with shorter output lengths may in fact be suboptimal. Therefore, to handle demand hybridity of LLM inferences, we need to jointly consider their resource consumptions in both compute and memory dimensions.

To summarize, due to demand uncertainty and hybridity, existing LLM schedulers fail to achieve high overall efficiency. We next devise a novel scheduler for efficient LLM scheduling, with optimizations respectively in output-length prediction, cost modeling and request queuing.

---

blocking, TTLT-friendly schedulers can naturally improve TTFT; for TPOT—which is defined [48, 13] as *dividing TTLT by the output token length*—reducing TTLT would proportionally improve TPOT. We will confirm them in evaluation.

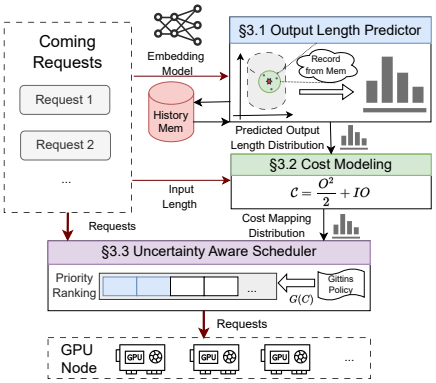
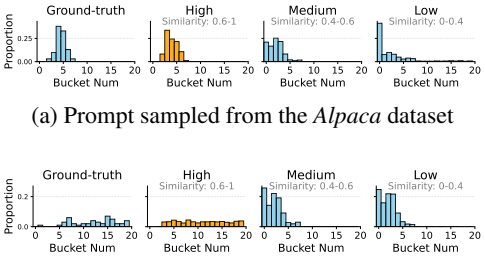


Figure 3: Overview of *SageSched* workflow.



(a) Prompt sampled from the *Alpaca* dataset

(b) Prompt sampled from the *Write* dataset

Figure 4: Output-length distribution of a request can be better approximated by historical requests with a higher prompt similarity.

3 SageSched Design

In this paper, we design *SageSched*, a novel LLM scheduler that, by properly handling demand uncertainty and hybridity, attains efficient TTLT performance. The overall workflow of *SageSched* is elaborated in Fig. 3. For each incoming LLM inference request, *SageSched* first predicts its output-length distribution with a light-weight yet also accurate predictor (Sec. 3.1). It then quantifies the overall service cost of that request based on the predicted output length, with both the memory and compute resources considered (Sec. 3.2). Finally, *SageSched* employs an uncertainty-aware scheduling algorithm to determine the runtime request queuing order, which can yield the optimal efficiency (Sec. 3.3). The source code is included in the supplementary material.

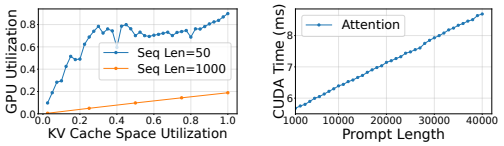
3.1 Semantic-aware History-based Predictor

As elaborated in Sec. 2.2, given the built-in uncertainty of LLM decoding process, a truly accurate prediction method must yield a *distribution* rather than a *number* when depicting the predicted output token length. In that regard, a straightforward solution is to customize the output-projection layers in existing prediction models (e.g., DistilBERT [20]) for distribution prediction; such a customized model essentially seeks to—given the semantic input prompt—directly *approximate* the ultimate generation effect. Yet, maintaining such a prediction model is still *costly* (due to the training overheads) and *inaccurate* (due to the complexity to approximate the inference effect).

In fact, given an inference prompt, we find its output length should be better predicted by *referring to similar requests in the recent history*—instead of by directly emulating the inner generation process with a fine-tuned model. Given the complexity of auto-regressive generation, it would be much easier to *evaluate prompt similarity* than to *directly emulate the generation effect*. Meanwhile, existing works [17] have revealed that, *the overall distributions of request output length within adjacent sliding time windows are relatively stable*, indicating high reference value of the recent serving history. Therefore, by selecting the past requests with a similar prompt, we can potentially make *easier-yet-more-accurate* demand prediction.

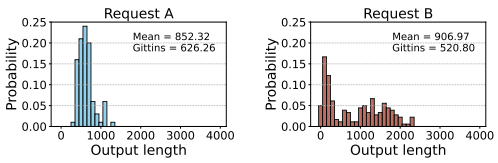
To explore, we empirically study the correlation between prompt similarity and output-length similarity. We sample two prompts respectively from the Alpaca and Write datasets (elaborated in Sec. 2.1). For each prompt, we first serve it 100 times, obtaining an output-length distribution as the ground-truth to predict. Then we classify the historical requests into three categories based on their similarity to the selected prompt; here similarity is quantified as the *cosine similarity* between two requests' prompt embeddings. As shown in Fig. 4, for each case, there exists a clear correlation between the prompt similarity and the output-length-distribution similarity: the curve shapes of *higher-similarity* buckets are *closer* to the target distribution.

Given the above analysis, we propose *semantic-aware history-based predictor*, which combines the *prompt contents* with the *historical execution status* to directly predict the output-length distribution. Specifically, we record the inference information (prompt contents and output lengths) of recently-



(a) Resource bound (b) Attention cost

Figure 5: Measurements on the instantaneous resource bound, as well as the compute cost characteristics in decoding. All measurements are conducted on an H800 GPU with Qwen3-32B model.



(a) Request-A (b) Request-B

Figure 6: An example illustrating the deficiency of mean-value-based request prioritization. While request-B has a larger mean value, due to its irregular demand-distribution shape, serving it first can be expected to behave better in average TTLT.

served requests. For each incoming request, we search[4] for those requests with a sufficiently-similar prompt (with a similarity threshold defaulted to 0.8), and use their output-length distribution as the prediction result. Such a method does not require fine-tuning any model, thereby being generally applicable to diverse LLMs; neither does it require emulating the complex generation process. In this way, we can make light-weight and also accurate prediction for output-length distribution.

3.2 Resource-bound-based Cost Modeling

With the above prediction method, we can now obtain the output-length distribution for each upcoming request. Nonetheless, as explained in Sec. 2.2, due to demand hybridity, it is deficient to directly use the output token length as the service cost of an LLM request. In this part we need to find a comprehensive cost-depicting model with both compute and memory resources considered.

We note that for multi-resource cost modeling, it is crucial to first understand which resource is the true bottleneck, which determines the follow-up cost analysis roadmap. In reality, an LLM service backend may be either *compute-bound* or *memory-bound*—depending on the runtime workload characteristics. Specifically, in Fig. 5(a) we depict[5] the relationship between compute and memory consumptions for a decode step served atop an H800 GPU with increasing batch size. We respectively fix the sequence length (including both the input length and the latest output length so far) to 50 and 1000. Longer sequences tend to be more memory-intensive: when serving short sequences, the GPU utilization is already very high before reaching the memory limit; yet for long sequences, the GPU utilization has not yet ramped up when GPU memory is full. Note that the service cost on a resource type matters only when that resource becomes the bottleneck. We therefore explore how to analyze the LLM service cost respectively in each status.

We first focus on inference cost modeling in *memory-bound* status. In such circumstances, the KVCache occupation of an LLM request is a key factor in cost modeling—less KVCache consumption allows more requests to be concurrently served (with low GPU utilization, serving more requests concurrently won't increase the iteration time). Therefore, in memory-bound scenarios, we need to primarily consider a request's cumulative KVCache consumption during its inference process. That is, let I and O be the input (i.e., prompt) and output token lengths of an inference; then its total service cost $\mathcal{C}$ can be depicted by $\mathcal{C}_M = \sum_{l=I}^{I+O} l \cdot U_{MT} \approx (\frac{O^2}{2} + IO)U_{MT}$, where U_{MT} represents the unit *memory-time* product (i.e., the KVCache space occupied by a single token in a single step).

We next turn to the *compute-bound* status. In such cases, the KVCache occupation of an LLM inference is not a concern on the service backend, and we only need to quantify its total computation amount over its lifetime. To that end, in Fig. 5(b), we measure the per-iteration attention-computation time (a key computing module in Transformers) in different decoding[6] steps when an inference

[4]Our history window has a size of 10,000 records and keeps updating in a *FIFO* manner. In cases where the high-similarity requests are insufficient (e.g., in the warm-up phase), we augment the search set with the requests from public datasets. We use the efficient *FAISS IndexFlat* [32] tool to perform embedding search, which in general takes less than 1 ms.

[5]In Fig. 5, GPU utilization is estimated by comparing the achieved TFLOPS to the peak TFLOPS, and the CUDA time is measured with `torch.profiler`.

[6]In this paper we focus on resource consumption in *decoding* steps; with the wide adoption of *reasoning LLMs* [4], the decoding phase of an inference often dominates its TTLT. Moreover, apart from the linear attention-computation time, there also exists a *constant* FFN computation time (as well as some minor time factors like CUDA kernel-launching delay) that is

is served alone. It confirms that the duration is linear in the accumulated sequence length so far. Therefore, we can depict the total computation cost of an inference as $\mathcal{C}_C = \sum_{l=I}^{I+O} l \cdot U_{CT} \approx (\frac{O^2}{2} + IO)U_{CT}$, where U_{CT} represents the unit *compute-time* product (i.e., the average computation amount required to process a single KVCache token in a single step).

Given the above analysis, we find that both the compute and memory costs share the same paradigm (although with different units, which does not affect the relative cost order among requests). Therefore, we can adopt a uniform cost modeling method in practice (with no need to switch on the bottleneck pattern): $\mathcal{C} = \frac{O^2}{2} + IO$. Despite such cross-resource paradigm consistency, we note that our formulation is substantially different from existing ones—e.g., purely O in [38, 13, 41], or a simple weighted summation of I and O in [42]. Our later evaluations in Sec. 4.3.2 confirm the superiority of such a modeling approach.

3.3 Uncertainty-aware Request Scheduling

With the above output-prediction and cost-modeling methods, we can now depict the cost distribution of each LLM request upon its arrival (which can mitigate—yet not eliminate—demand uncertainty). Here we need to design an uncertainty-aware scheduling algorithm that can attain the theoretically optimal TTLT performance.

In request queuing, we need to calculate a proper scheduling index out of each request's cost distribution (for which the mean value is not optimal, as suggested in Fig. 6). In fact, we notice that selecting a highest-priority inference to serve is similar to selecting the most-rewarding arm to take in a *multi-armed bandit problem* [31]: the cost distribution is essentially a reward distribution depicting how allocating additional resources to a request can help to yield inference completion. For such a multi-armed bandit problem with nondeterministic reward but deterministic reward distribution, in the literature it is known that the *Gittins policy* [16] can achieve the theoretically optimal performance.

Specifically, given a request with a cost distribution of $\mathcal{D}$, the Gittins policy calculates its Gittins index defined as $G(\mathcal{D}) = \inf_{\Delta>0} \frac{\mathrm{E}[\min\{X,\Delta\}]}{\mathrm{P}(X\leq\Delta)}$ with $X \sim \mathcal{D}$; requests with smaller Gittins indices are prioritized in scheduling. In that formula, $\mathrm{E}[\min\{X, \Delta\}]$ represents the request service consumption when allocated a service budget Δ, and $\mathrm{P}(X \leq \Delta)$ represents its probability of successful completion before Δ is used up; G thus represents the minimum amortized cost ever attainable for that request. It has been proven that scheduling jobs based on such Gittins indices (instead of the distribution expectation values) can yield the minimum average latency (TTLT) [15]. The effect can be observed in Fig. 6, where the Gittins policy prioritizes the request that is more likely to complete soon.

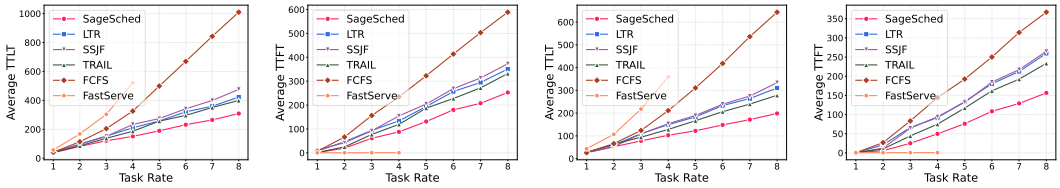
Furthermore, the Gittins index of each pending request shall be properly updated at runtime after inference commencement. As indicated by TRAIL [41] and FastServe [48], enabling preemption is necessary to optimize the average TTLT (with negligible overheads given recent techniques like *swap-compute overlapping* and *KVCache prefetching*). Ideally, we need to keep refreshing the Gittins index of each active inference at runtime—with the information of *remaining* service cost distribution. Yet, continuously doing so after each inference step is still impractical—due to the high monitoring overhead as well as the thrashing risk (i.e., the relative scheduling order of two requests with similar Gittins indices may frequently reverse). We therefore divide each request's cost range into multiple (defaulted to 10) buckets; the Gittins index of each request is refreshed only at bucket boundaries. In this way, we can seek a proper balance between rescheduling timeliness and system stability.

4 Evaluation

4.1 Setup

Hardware platform. In our testbed experiments, we respectively use Llama3.1-8B [3] and Qwen3-32B [39] as the LLM backend; the Llama3.1-8B model is deployed on a server equipped with an A40-PCIe-48GB GPU, and the Qwen3-32B model is deployed on a server with an H800-PCIe-96GB GPU. Moreover, we also build a simulator of up to 64 GPUs to evaluate the scalability

independent of the sequence length. However, the FFN time can be remarkably amortized in long decodes or with large batch sizes (given the GPU GEMM optimizations); so for simplicity we skip it here. Note that we have already included the memory I/O overhead into the compute time (which is the end-to-end time containing HBM-SRAM loading and kernel execution).



(a) Llama3.1-8B TTLT (b) Llama3.1-8B TTFT (c) Qwen3-32B TTLT (d) Qwen3-32B TTFT

Figure 7: End-to-end performance on mixed datasets.

performance of *SageSched*. Moreover, we also evaluate *SageSched* in a heterogeneous multi-instance deployment with 2 A40-PCIe-48GB GPUs and 2 RTX 4090-24GB GPUs; the results are presented in Appendix A.2.

Workloads. In our experiments we use three typical inference datasets as in [19]: ShareGPT [22], Alpaca-Summarization [21], and Document-Write [23]. The characteristics of those datasets are previously elaborated in Fig. 1(b). Regarding request submission interval, we employ a Poisson distribution under different λ hyperparameters (yielding diverse task rates). We further validate the effectiveness of *SageSched* using the real-world BurstGPT trace [47]. Detailed experimental settings and results are provided in Appendix A.1.

Baselines. We compare our *SageSched* solution with multiple representative baselines: FCFS [25], FastServe [48], SSJF [38], LTR [13], and TRAIL [41]. All of those baselines have been elaborated in Sec. 2.2, and their hyperparameters are all set as default. Besides, by default, for *SageSched* the similarity threshold in the predictor is set to 0.8, and the bucket size for Gittins analysis is set to 200.

Metrics. As elaborated in Sec. 2.1, we use the average TTLT as the primary scheduling metric. That said, we also evaluate the average TTFT performance of each scheduler. We put results on other non-essential metrics (e.g., P90 TTLT, P90 TTFT and avg. TPOT) in Appendix A.1.

4.2 End-to-end Experiment

Experiments with requests from mixed datasets. In our end-to-end evaluation, we first merge the three datasets together, the requests from which are then randomly submitted; we experiment respectively under different task rates. As shown in Fig. 7, *SageSched* demonstrates the best scheduling efficiency in TTLT across all the setups. For example, when inferences are submitted at a rate of 8 requests-per-second (RPS) to the Qwen3-32B model, *SageSched* surpasses the second-best scheduler, TRAIL, by 28.7%. Such improvements are higher in cases with more intensive competition, because *SageSched* can consistently work out the best scheduling order by properly handling the uncertainty and hybridity characteristics of LLM inference demands.

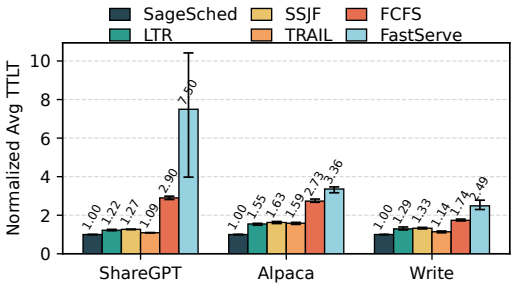


Figure 8: End-to-end performance on each dataset.

Regarding TTFT, while it is not the primary focus of this paper, Fig. 7 indicates that *SageSched* can also attain a satisfying performance, because it can effectively mitigate the head-of-line blocking problem. In particular, although FastServe—by always prioritizing the newly-arrived requests—can attain the best TTFT, it behaves much worse than other schedulers in TTLT due to its interleaved execution mode with MLFQ. In that sense, *SageSched* is the only scheduler that can behave well in both TTLT and TTFT.

Experiments with requests respectively from each dataset. Recall that the three datasets we use have different input-output length distributions (Fig. 1(b)); we experiment respectively with each dataset, as shown in Fig. 8 (the error bars show the outliers). We first note that *SageSched* consistently

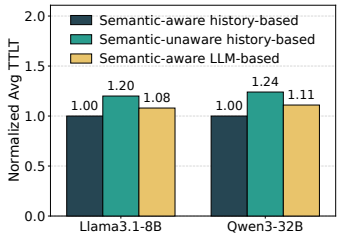


Figure 9: Performance comparison of different output-length prediction methods.

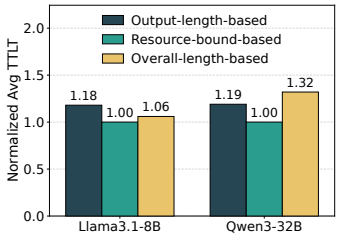


Figure 10: Performance comparison of different service cost modeling methods.

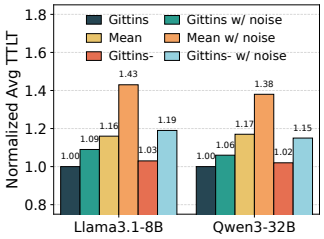


Figure 11: Comparison of different scheduling methods, with and without added cost noise.

performs the best for each dataset. In particular, its performance superiority peaks for the Alpaca (Summarization) dataset: this is because requests in the Alpaca dataset typically have long input lengths, for which existing output-length-based scheduling methods are highly inaccurate in cost modeling.

4.3 Microscopic Deep Dive

In this part, we dive deep into the effectiveness of each solution component in *SageSched*. Unless otherwise specified, we use the same setup as in Fig. 7 (under an RPS of 8).

4.3.1 Superiority of Our Predictor Design

Given the deficiency of conventional output-length predictors, in Sec. 3.1 we propose a novel *semantic-aware history-based predictor*. Here we check its necessity with regard to the end-to-end TTLT performance. Specifically, we incorporate two baselines representing the other prediction choices: (1) *semantic-unaware history-based predictor*—which predicts the output length of a request by referring to the historical requests with a similar input length (following the same filtering method as ours), and (2) *semantic-aware LLM-based predictor*—still using a DistilBERT model as in [38] yet, to predict a distribution, we remove the `logits.argmax` layer. In Fig. 9, we show the performance of *SageSched* respectively with different predictors. It shows that with our semantic-aware history-based predictor, due to its high distribution-prediction accuracy, *SageSched* achieves the best TTLT performance.

Moreover, we note that our predictor also surpasses others in training and prediction overheads. First, our method completely eliminates training overhead, whereas for the LLM-based predictor, it takes several hours to construct a training dataset and also tens of minutes to perform model fine-tuning. Second, the average per-request prediction latency is below 0.5 ms under our predictor (consisting of 0.22 ms for semantic embedding extraction and 0.15 ms for vector retrieval), yet for the LLM-based predictor it is approximately 3.6 ms. Those results above jointly confirm that our semantic-aware history-based predictor is light-weight and also accurate.

4.3.2 Superiority of Our Cost Modeling Method

In Sec. 3.2, after analyzing both the compute- and memory-bound cases, we developed a *resource-bound-based modeling* method to depict LLM inference cost. To confirm its necessity, we respectively replace it with two other methods: (1) *output-length-based modeling*—which directly uses the output length as the inference cost [41, 38], and (2) *overall-length-based modeling*, which combines the input and output lengths as the cost (we assign the output length a doubled weight as in [42]). Fig. 10 shows the overall TTLT performance under different modeling methods. It confirms the necessity to adopt the resource-bound-based method for cost modeling.

4.3.3 Superiority of Our Scheduling Policy

Recall that our uncertainty-aware scheduling algorithm in Sec. 3.3 has two key techniques: *Gittins-index-based ordering*, and *runtime Gittins-index refreshing*. Here we respectively evaluate the benefit of each technique; besides, we also need to check the robustness of scheduling quality in cases

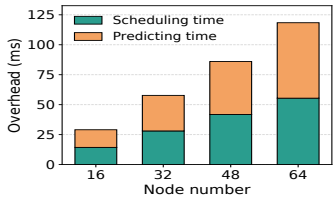
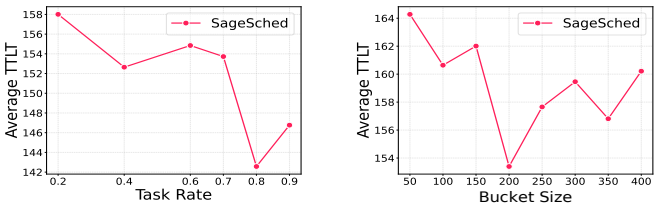


Figure 12: Predicting and scheduling overheads of *SageSched* at different scales.



(a) Predictor hyperparameter (b) Scheduler hyperparameter

Figure 13: Sensitivity analysis of *SageSched* hyperparameters.

with inaccurate predictions. Specifically, we first introduce two additional baselines: (1) *Mean*—which follows an SRPT-style scheduling policy by ordering requests according to the *mean* value of their cost distributions, and (2) *Gittins*−—which applies the Gittins policy but without timely refreshing the Gittins index at bucket boundaries. Moreover, for each scheduler, we also measure its performance against *inaccurately-predicted* cost distributions (by mixing a *uniform* distribution into the original distribution following a weight ratio of 1:4). As shown in Fig. 11, both Gittins-index-based ordering and runtime Gittins-index refreshing are beneficial for TTLT performance; besides, the added noise incurs much less performance degradation to our Gittins-index-based algorithm than to other baselines.

4.4 Overhead and Sensitivity Analysis

In this part we study the scalability issue of *SageSched* and also perform sensitivity analysis on the hyperparameters introduced. Unless otherwise specified, we also use the same setup as in Fig. 7.

Scalability performance. To evaluate the scalability performance of *SageSched*, we simulate large-scale cluster deployments with up to 64 GPU nodes. We retain the frontend prediction and cost-modeling methods, together with the backend Gittins-index computation and scheduling logic. The system load (RPS) is increased proportionally with the cluster size. On average, each node serves 8 RPS, with up to 1,000 requests buffered in the queue. We measure the per-request latency incurred during the predicting and scheduling stages, assuming a fixed output length of 1,000 tokens. As shown in Fig. 12, the latency grows only linearly as the cluster scales. Even at the largest scale of 64 nodes, the average additional latency per request remains around 100 ms. Given that typical LLM inference workloads operate at the granularity of several seconds to minutes (as in Fig. 7), this overhead is indeed negligible. Moreover, for large clusters, we can deploy multiple concurrent schedulers to further alleviate such overhead.

Similarity threshold. In Sec. 3.1 we adopt a cosine-similarity threshold to filter out the historical requests with similar prompts. We respectively set it to different values, with performance shown in Fig. 13(a). The results confirm that setting it to 0.8 yields the best performance—an over-large threshold renders the selected requests less abundant (compromising the capturing of demand uncertainty), yet an over-small threshold would pollute the sampled distribution with irrelevant requests.

Bucket size. In Sec. 3.3 we adopt a bucket-size hyperparameter to control the refreshing frequency of the Gittins index. As in Fig. 13(b), we measure the avg. TTLT under *SageSched* with different bucket sizes. It suggests that a bucket size of 200 can yield the best performance (which we thus adopt as default): smaller bucket sizes lead to more priority calculation and re-scheduling overheads, yet larger bucket sizes compromise the accuracy of priority assignment; both degrade the overall efficiency.

5 Conclusion

In this work, we present *SageSched*, an efficient scheduler tailored for the uncertainty and hybridity characteristics of LLM workloads. *SageSched* introduces a semantic-aware history-based predictor to predict output-length distribution in a light-weight and also accurate manner. It also models the

overall service cost of LLM inferences with both memory and compute bounds considered. Besides, *SageSched* adopts an uncertainty-aware scheduling policy which leverages the Gittins index for the best queuing performance. Testbed and simulation experiments confirm that *SageSched* can substantially improve the end-to-end efficiency performance, with a TTLT reduction of over 28.7%.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC), under Grant No. 62572304.

References

[1] E Abrossimov, Marc Rozier, and Marc Shapiro. Generic virtual memory management for operating system kernels. In *Proceedings of the twelfth ACM symposium on Operating systems principles*, pages 123–136, 1989.

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

[3] Meta AI. Llama-3.1-8B-Instruct. `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`, 2024. Accessed: 2026-01-29.

[4] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-KV: Redundancy-aware KV cache compression for reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[5] Guojun Chen, Xiaojing Yu, Neiwen Ling, and Lin Zhong. TypeFly: Flying drones with large language model. *arXiv preprint arXiv:2312.14950*, 2023.

[6] Cursor. Cursor. `https://cursor.com/`, 2024. Accessed: 2026-01-27.

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

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

[9] DeepSeek. DeepSeek. `https://www.deepseek.com/`, 2024. Accessed: 2026-01-27.

[10] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.

[11] Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. MuxServe: Flexible spatial-temporal multiplexing for multiple LLM serving. In *International Conference on Machine Learning (ICML)*. OpenReview.net, 2024.

[12] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[13] Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient LLM scheduling by learning to rank. *Advances in Neural Information Processing Systems*, 37:59006–59029, 2024.

[14] GeeksforGeeks. Comparison of different CPU scheduling algorithms in operating systems. `https://www.geeksforgeeks.org/comparison-of-different-cpu-scheduling-algorithms-in-os/`, 2026. Accessed: 2026-01-27.

[15] John C Gittins. Multi-armed bandit allocation indices. *Wiley-Interscience Series in Systems and Optimization*, 1989.

[16] John C Gittins and David M Jones. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979.

[17] Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. Past-future scheduler for LLM serving under SLA guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 798–813, 2025.

[18] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024.

[19] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, et al. Inference without interference: Disaggregate LLM inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024.

[20] Hugging Face. DistilBERT. `https://huggingface.co/docs/transformers/model_doc/distilbert`, 2024. Hugging Face Transformers Documentation; Accessed: 2026-01-27.

[21] Hugging Face Datasets. Alpaca-PubMed summarization dataset. `https://huggingface.co/datasets/ZhongshengWang/Alpaca-pubmed-summarization`, 2025. Accessed: 2026-01-27.

[22] Hugging Face Datasets. ShareGPT_Vicuna_unfiltered dataset. `https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered`, 2025. Accessed: 2026-01-27.

[23] Hugging Face Datasets. write_doc_sft_v1 dataset. `https://huggingface.co/datasets/lancexiao/write_doc_sft_v1`, 2025. Accessed: 2026-01-27.

[24] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S^3: Increasing GPU utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

[25] 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.

[26] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 155–172. USENIX Association, 2024.

[27] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023.

[28] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding. *CoRR*, abs/2310.07177, 2023.

[29] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning (ICML)*. OpenReview.net, 2024.

[30] Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016.

[31] Aditya Mahajan and Demosthenis Teneketzis. Multi-armed bandit problems. In *Foundations and applications of sensor management*, pages 121–151. Springer, 2008.

[32] Meta AI. FAISS: A library for efficient similarity search and clustering of dense vectors. `https://ai.meta.com/tools/faiss/`, 2024. Accessed: 2026-01-28.

[33] Meta AI. Llama. `https://www.llama.com/`, 2024. Accessed: 2026-01-27.

[34] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating generative LLM serving with speculative inference and token tree verification. *CoRR*, abs/2305.09781, 2023.

[35] OpenAI. ChatGPT. `https://openai.com/index/chatgpt/`, 2024. Accessed: 2026-01-27.

[36] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 293–307, 2015.

[37] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *ACM Transactions on Storage*, 2024.

[38] 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.

[39] Qwen. Qwen3-32B. `https://huggingface.co/Qwen/Qwen3-32B`, 2025. Accessed: 2026-01-29.

[40] Pol G. Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll. Berral. Mind the memory gap: Unveiling GPU bottlenecks in large-batch LLM inference. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pages 277–287, 2025.

[41] 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.

[42] 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)*, pages 965–988, 2024.

[43] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–191. USENIX Association, 2024.

[44] vLLM Project. vLLM (v0.8.2). `https://github.com/vllm-project/vllm/tree/v0.8.2`, 2025. Accessed: 2026-01-27.

[45] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[46] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. LLM^3: Large language model-based task and motion planning with motion failure reasoning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12086–12092. IEEE, 2024.

[47] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5831–5841, 2025.

[48] 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.

[49] Yuxing Xiang, Xue Li, Kun Qian, Yufan Yang, Diwen Zhu, Wenyuan Yu, Ennan Zhai, Xuanzhe Liu, Xin Jin, and Jingren Zhou. Aegaeon: Effective GPU pooling for concurrent LLM serving on the market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP '25)*, 2025.

[50] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[51] 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.

[52] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[53] 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.

[54] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: an LLM-empowered LLM inference pipeline. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

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

A Appendix

A.1 Additional Experimental Results

Table 1: Average TTLT on the BurstGPT trace.

Workload	Avg. Interval	SageSched	FCFS	SSJF
Low-Intensity	6.26s	25.40	25.50	25.51
High-Intensity	0.14s	272.83	894.36	420.05
Overall	–	159.53	306.05	207.02

BurstGPT trace performance. Regarding request arrival pattern, we evaluate *SageSched* with the request arrival-time records from the realistic BurstGPT trace [47], which has time-varying arrival intensities. As shown in Tab. 1, we measure the TTLT performance over the entire trace and also over a sampled low-intensity/high-intensity period. The results show that the *SageSched* benefit vanishes during the low-intensity period (because scheduling order does not matter when few requests are backlogged), yet remarkably amplifies during the high-intensity period.

Table 2: Robustness of scheduling performance under different prediction noise ratios.

Noise Ratio	0	0.1	0.2	0.4
Avg. TTLT (Gittins-index)	153.72	158.33	167.55	195.23
Avg. TTLT (Mean-index)	172.17	182.93	199.84	212.13

Robustness to prediction noise. We further measure performance robustness as shown in Tab. 2 by changing the noise ratio from 0.2 (default value in Fig. 11) to 0, 0.1 and 0.4, respectively. The overall TTLT results show that, while a larger prediction error prolongs the average TTLT, the Gittins-index-based scheduling of *SageSched* is consistently better than the mean-index-based one in each case.

Table 3: Performance comparison of additional metrics under different task rates.

Task Rate	Algorithm	P90 TTLT (s)	P90 TTFT (s)	Avg. TPOT (s)
2	SageSched	247.33	78.63	0.28
	FCFS	234.92	167.95	2.39
	SSJF	250.95	118.18	0.44
	LTR	250.97	121.56	0.39
	TRAIL	244.36	84.37	0.31
4	SageSched	484.35	307.22	0.88
	FCFS	512.51	433.92	23.00
	SSJF	524.09	289.93	2.08
	LTR	491.90	322.54	1.48
	TRAIL	465.73	312.89	1.17
6	SageSched	759.33	544.05	1.33
	FCFS	888.73	692.88	43.90
	SSJF	779.77	581.96	3.95
	LTR	758.86	619.01	2.32
	TRAIL	700.02	561.35	1.86
8	SageSched	1036.89	742.83	1.79
	FCFS	1209.83	1120.61	73.30
	SSJF	1095.76	766.68	8.92
	LTR	1013.27	783.68	2.98
	TRAIL	1059.34	745.12	2.48

Table 4: Prediction overhead under different prompt lengths.

Prompt Length	512	1k	2k	3k	4k
Overhead (ms)	0.37 ± 0.03	0.62 ± 0.05	1.18 ± 0.08	1.84 ± 0.09	2.55 ± 0.13

Table 5: Prediction overhead under different history window sizes.

History Window Size	10k	100k	1M
Overhead (ms)	0.37	1.29	10.39

Additional metrics. As shown in Tab. 3, we further provide the performance results on P90 TTLT, P90 TTFT and TPOT (calculated following the method in Footnote 3) for the Llama3.1-8B experiment in Fig. 7(a)–(b). It shows that *SageSched* achieves the lowest average TPOT under all task rates and the lowest P90 TTFT in most cases, while its P90 TTLT stays within 9% of the best baseline.

Factors affecting prediction overheads. As shown in Tab. 4 and Tab. 5, we measure the prediction overheads with very large prompt lengths and window sizes. On the one hand, the prediction overheads do increase with the two factors; on the other hand, the absolute overheads are still modest. (By contrast, given a prompt length of 512 tokens, the DistilBERT-based prediction method as adopted in SSJF has a prediction overhead of 3.6 ± 0.3 ms.)

A.2 Extension to Multi-instance Serving

We further extend *SageSched* to the multi-instance serving scenario, where multiple backend instances serve the same model. Before routing, *SageSched* first predicts the output-length distribution for each incoming request, models its resource-bound-based cost, and computes a global priority order across all pending requests. This priority order is shared by all backend instances, i.e., the relative ordering of requests is independent of the specific instance to which they may be dispatched.

The router then dispatches requests according to this global priority order. Specifically, when the highest-priority request can be admitted by at least one backend instance, the router dispatches it to an eligible instance for execution. An instance is considered eligible if it has sufficient remaining execution capacity and KVCache space to admit the request. To select among multiple eligible instances, the router ranks instances according to $\alpha = \frac{C}{M}$, where C denotes the average computation time of a single iteration of the corresponding instance, and M is the total KVCache capacity of the corresponding instance. Intuitively, α characterizes the relative significance of the resource cost on the current instance: a smaller α indicates that the request is less likely to impose severe resource pressure on that instance. Therefore, when multiple instances can execute the same request, *SageSched* preferentially dispatches it to the instance with the smaller α. If several eligible instances have identical resource configurations, *SageSched* further applies load balancing among them.

We deploy four Llama3-8B instances on a heterogeneous testbed with two A40 GPUs and two RTX 4090 GPUs, and evaluate TTLT under different request rates. As shown in Table 6, *SageSched* consistently outperforms FCFS+load-balancing and SSJF+load-balancing across all RPS settings. The improvement comes from *SageSched*'s ability to jointly account for request-level resource costs and instance-level heterogeneity in serving capability.

A.3 A Full Version of Related Work

Our work focuses on *request-level scheduling* under uncertain and hybrid (compute–memory) demands; the following lines of research are largely orthogonal and can be combined with *SageSched*.

LLM inference acceleration. A substantial body of work reduces the *per-token* computation and memory cost of LLM inference. At the kernel level, FlashAttention and FlashAttention-2 reorganize attention computation to improve memory locality and reduce redundant reads/writes, yielding faster and more memory-efficient exact attention [8, 7]. Model compression and low-bit quantization further reduce memory footprint while retaining accuracy, e.g., INT8 matrix multiplication and post-training

Table 6: Avg. TTLT under different request rates in the multi-instance setting.

Algorithm	RPS			
	8	16	24	32
SageSched	89.72	161.25	237.28	311.54
FCFS + Load Balancing	117.86	355.49	756.83	1217.30
SSJF + Load Balancing	94.62	243.35	389.11	537.81

quantization for transformer weights [10, 12]. Recent decoding-side acceleration methods exploit *speculative execution*: speculative decoding uses a fast draft model to propose multiple tokens and then verifies them with the target model [27], while subsequent systems improve speculative serving efficiency via token-tree verification and online mechanisms [34, 28].

Beyond token-level acceleration, system designs optimize the end-to-end pipeline. Prefill–decode disaggregation decouples throughput-oriented prefill from latency-critical decoding and restructures KVCache placement across devices [55, 37, 18, 19], while chunked-prefill overlaps long-context prefill with decoding and improves cache reuse to better manage throughput–latency trade-offs [2, 19].

These techniques improve the *execution efficiency* of each request; *SageSched* is complementary by deciding *which* request to serve next given bounded resources and variable request lengths.

LLM serving architectures and multiplexing. Many systems improve backend utilization by redesigning batching, admission control, and multi-tenant resource sharing. Foundational serving frameworks (e.g., Orca-style continuous batching and modern engines such as vLLM and SGLang) emphasize high device utilization and efficient KVCache management [51, 25, 53]. For multi-model and multi-tenant settings, recent work explores finer-grained multiplexing and pooling: MuxServe enables flexible spatial–temporal multiplexing to co-locate multiple LLM workloads on shared GPUs [11], and Aegaeon proposes effective GPU pooling for concurrent LLM serving on the market [49].

These systems primarily address *resource sharing and placement* across models and tenants; *SageSched* instead targets *within-backend* request ordering under uncertain service times, and can be integrated as the per-GPU (or per-pool) scheduler.

LLM request scheduling and fairness. A growing literature studies request scheduling to mitigate head-of-line blocking and improve tail latency. Production engines often start with FCFS, which is simple but can severely degrade TTLT under heterogeneous request lengths [25, 53]. FastServe approximates SRPT using an MLFQ-like policy with periodic priority demotion, trading latency gains for additional preemption overheads [48]. More recent schedulers adopt *predict-and-order*: S^3 uses a fine-tuned small model to estimate generation length [24], PO queries an LLM to anticipate its output length [54], SSJF predicts output length with a proxy model to approximate SJF [38], and learning-to-rank approaches predict relative length ranks rather than exact lengths [13]. Embedding-based schedulers further exploit prompt features extracted from LLM layers to improve prioritization [41]. In parallel, fairness-aware serving formalizes the tension between efficiency and service differentiation and proposes scheduling mechanisms to meet fairness objectives [42]. Llumnix addresses dynamic scheduling by incorporating runtime conditions and enabling adaptive reordering for better service quality [43].

Compared to these works, *SageSched* differs in three aspects: (i) it targets *distributional* (rather than point) perception of output length, avoiding fragile single-value predictions; (ii) it models *hybrid* compute–memory costs induced by KVCache growth, rather than using output length alone as the service cost; and (iii) it explicitly incorporates IO impact into preemption decisions, aiming to avoid excessive swapping while still correcting priority.

KVCache management and long-context serving. Since decoding repeatedly attends to all previous tokens, KVCache is often the dominant memory bottleneck. PagedAttention (as implemented in vLLM) improves KVCache allocation by paging KV blocks to reduce fragmentation and improve memory efficiency [25]. Beyond allocation, systems reduce KVCache *size* via compression or selective retention: attention sinks enable efficient streaming by stabilizing attention patterns [50],

and heavy-hitter-based methods retain the most important tokens to approximate full attention under memory constraints [52]. More recent work dynamically manages KVCache across time: InfiniGen proposes dynamic KVCache management for efficient generative inference [26], while KIVI quantizes the KVCache asymmetrically to 2-bit without tuning, reducing memory pressure and bandwidth [29].

These advances improve the feasible concurrency and per-step efficiency; *SageSched* is orthogonal and can further improve TTLT by prioritizing requests using cost distributions that implicitly reflect KVCache growth and resource contention.

A.4 Limitations

SageSched improves LLM serving efficiency by leveraging historical request information to predict output-length distributions and by scheduling requests according to resource-aware cost distributions. However, it has several limitations.

First, *SageSched* relies on the availability of representative recent history. Although our semantic-aware history-based predictor is training-free and can naturally adapt to different LLMs, its prediction quality may degrade during cold-start periods or under abrupt workload distribution shifts, where few semantically similar historical requests are available. In our implementation, we mitigate this issue by maintaining a sliding history window and augmenting the search space with public datasets during warm-up, but fully handling extreme distribution shifts remains future work.

Additionally, *SageSched* focuses on optimizing average end-to-end latency, especially Time-to-Last-Token (TTLT). While this metric is important for many agentic and API-style workloads, production serving systems may also require strict fairness, deadline, priority, or service-level-objective constraints across users and applications. Extending *SageSched* to jointly optimize efficiency and such policy constraints is an interesting direction for future work.

A.5 Broader Impact

This work aims to improve the efficiency of LLM serving systems. By reducing request completion latency and improving resource utilization, *SageSched* can potentially lower the operational cost and energy consumption of LLM services. This may make LLM-based applications more accessible, especially for latency-sensitive workloads such as agentic systems, code generation, and long-form generation.

At the same time, more efficient LLM serving may also reduce the cost of deploying large-scale generation systems, including applications that could produce low-quality, misleading, or harmful content. *SageSched* itself does not modify model capabilities, training data, or generation behavior; it only changes the scheduling order of inference requests in the serving backend. Therefore, its safe deployment should be combined with existing model-level safety alignment, content moderation, access control, and usage monitoring mechanisms.

We do not collect private user data or conduct human-subject studies in this work. The workloads used in our experiments are based on public datasets and traces, and the system is evaluated for performance and resource-efficiency purposes.

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 (Sec. 1) clearly state the paper's main claims: *SageSched* addresses demand uncertainty and hybridity in LLM inference scheduling via semantic-aware history-based prediction, resource-bound-based cost modeling, and Gittins-index-based uncertainty-aware scheduling. The claimed performance improvement is supported by testbed and simulation experiments reported in Sec. 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: We include a dedicated limitations section (Appendix A.4) discussing the dependence on representative serving history and the focus on average TTLT.

   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: [N/A]

Justification: The paper does not introduce new theoretical results or provide new formal proofs. It uses the known optimality property of the Gittins policy for jobs with unknown durations but known duration distributions, and cites the relevant prior work in Sec. 3.3.

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 describes the system implementation, hardware platforms, workloads, baselines, metrics, and major hyperparameters used in the experiments in Sec. 4. Sec. 3 also provides sufficient algorithmic details for reproducing the prediction, cost modeling, and scheduling pipeline.

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: We provide an anonymized code package and reproduction instructions in the supplemental material. The source code will also be publicly released upon acceptance.

   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 LLM backends, GPU platforms, workloads, request arrival processes, baselines, metrics, and default hyperparameters in Sec. 4. It also reports the training setup for learning-based prediction baselines and provides additional details for scalability simulations and sensitivity analysis (Sec. 4.4).

   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: Each experiment is repeated three times with different random seeds, and we report the mean value with one-standard-deviation error bars in Fig. 8.

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 GPU platforms used in the experiments, including A40-PCIe-48GB, H800-PCIe-96GB, and a heterogeneous multi-instance deployment with A40 and RTX 4090 GPUs in Sec. 4.

   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 focuses on improving the efficiency of LLM serving systems and does not involve human-subject experiments, private user data collection, or unsafe model training. We have reviewed the NeurIPS Code of Ethics and believe the work conforms to it.

   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: [Yes]

    Justification: We include a broader impact discussion. SageSched may reduce LLM serving latency, cost, and energy consumption, but more efficient serving may also lower the barrier for potentially harmful LLM applications. SageSched does not alter model capabilities or safety behavior and should be deployed together with model-level safety and usage-control mechanisms.

    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 new pretrained language model, image generator, scraped dataset, or other high-risk asset. SageSched is a serving-system scheduler and does not introduce new model-generation capabilities.

    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: The paper cites the existing models, frameworks, and datasets used in the experiments, including vLLM, Llama3.1-8B, Qwen3-32B, ShareGPT, Alpaca-Summarization, and Document-Write. They all comply with open-source licenses and can be used for scientific research.

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: [N/A]

Justification: The paper does not introduce a new dataset or pretrained model asset. The implementation will be released upon acceptance, and documentation can be provided together with the released code.

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 human-subject studies, user studies, or crowdsourcing experiments, so IRB approval 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: LLMs were used only for writing, editing, or formatting purposes.

    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.