ROUND-ROBIN SHORTEST JOB FIRST INFERENCE SCHEDULING FOR LLMS

Sreya Gogineni [*1] **Justin Paul** [*1] **Sarah Stec** [*1] **Brandon Zhang** [*1]

ABSTRACT

Serving large language models requires efficient scheduling of a diverse set of concurrent requests. We present a novel scheduling algorithm that combines approximate Shortest Job First (SJF) and Round-Robin (RR) scheduling strategies. By estimating prompt response lengths prior to scheduling, we can prioritize shorter jobs, preventing them from being starved by longer requests. Additionally, a Round-Robin processing strategy preempts longer-running tasks after generating a fixed number of tokens, enabling faster completion of shorter jobs and improving system responsiveness. The combination of these two approaches reduces average time-to-first-token (TTFT) with only a marginal increase in end-to-end latency (E2EL) which enhances user experience and improves fairness.

1 BACKGROUND AND MOTIVATION

Inference scheduling for large language models is foundational for supporting the wealth of chatbot services that are available today. Chatbots may receive millions of requests every day and organize compute resources hastily, so fairly serving every request quickly becomes a challenging problem.

While there are many different targets for optimization in the inference pipeline, this paper focuses on ranking requests for service priority to minimize user waiting times when the servers are faced with a high influx of requests.

Many popular LLM inference serving frameworks today, like vLLM (NVIDIA, 2024a) and Nvidia Triton (NVIDIA, 2024b), use the simplest scheduling strategy, **First-Come First-Serve (FCFS)**. While FCFS is easy to implement, has very little scheduling overhead, and avoids context-switching costs due to preemption, its tendency to serve long requests at the head of the line without interruption leads to service delays for waiting jobs.

A straightforward strategy to reduce head-of-line blocking is **Round-Robin (RR) scheduling**, which improves fairness and latency by periodically switching between requests. This strategy prevents starvation of later-incoming requests and allows shorter tasks to be completed earlier, even without prior knowledge of their lengths. However, the frequent job switching inherent to RR can introduce significant overhead. For LLM inference, this overhead stems from handling the Key-Value (KV) cache (Cheng et al., 2024): pausing a request typically requires either swapping the KV cache to host memory (and later swapping it back) or discarding and recomputing it when the request resumes.

The **Shortest Job First (SJF)** algorithm is theoretically optimal for minimizing average time-to-first-token (TTFT) and end-to-end latency (E2EL) but comes with a practical limitation: it requires knowledge of job durations in advance. In the context of LLM inference, predicting request processing time is inherently difficult due to the autoregressive nature of LLMs. Many techniques have been developed to predict output length, such as by training a proxy model that takes in the prompt and returns an output-length prediction (Qiu et al., 2024; Zheng et al., 2024b; Abgaryan et al., 2024; Hu et al., 2024), but they are not always accurate. The true length of the response—and, consequently, the total processing time—remains uncertain until the generation is complete. Performing output-length prediction before or during each request service also has added time and resource costs to consider.

These different techniques work well for separate Quality-of-Experience (QoE) metrics. RR works well for reducing the TTFT metric but introduces switching overhead that increases overall E2EL. SJF improves both median TTFT and E2EL, but can starve longer jobs from getting service and relies on an accurate and low cost job length predictor. We aim to balance the strengths and weaknesses of these algorithms through our novel **Round-Robin Shortest Job First (RR-SJF)** algorithm.

RR-SJF preempts running decodes after a host-specified number of tokens are generated and then services waiting requests. Waiting requests are ordered for service first by the amount of time they've spent on the waiting queue, imitat-

[*]Equal contribution [1]University of Michigan, Ann Arbor, Michigan, USA. Correspondence to: Sreya Gogineni <gosreya@umich.edu>, Justin Paul <justpaul@umich.edu>, Sarah Stec <sastec@umich.edu>, Brandon Zhang <bdwzhang@umich.edu>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2024.

ing least attained service to prevent request starvation, and second according to shortest speculated running times, approximating SJF. Requests are then processed incrementally via RR according to this ordering.

2 METHOD

2.1 Round-Robin Shortest Job First

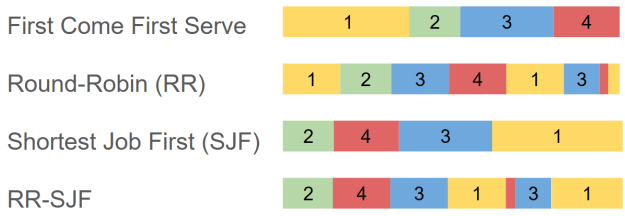

Figure 1. RR-SJF compared to other scheduling algorithms. Each rectangle represents the time the corresponding request number is processed.

RR-SJF combines SJF's and RR's improvements to latency and fairness. It achieves this by specifying the order in which requests are incrementally processed via RR, running the shortest requests first within each round and the longest requests at the end of each round.

To illustrate, **Figure 1** depicts the relative ordering and time spent processing 4 requests sent concurrently to the server. Both FCFS and RR process requests in the same order they come in (1,2,3,4), where FCFS processes each request to completion and RR periodically preempts requests after a fixed time slice. In SJF, we observe requests are processed to completion in order of ascending job length (2,4,3,1).

In RR-SJF, requests are ordered like SJF, by ascending job length (2,4,3,1). Request 2 is first run to completion because it did not exceed its allocated time slice. RR-SJF then processes the remaining requests in a Round-Robin manner, preempting requests when they exceed their corresponding time slices. This is why we observe RR-SJF process requests 4,3, and 1 sequentially for the same fixed period of time, before starting another round to finish processing these same requests. Note that after requests 2,3,4 have finished, request 1 is the only one remaining, so it can run to completion uninterrupted despite extending beyond one time slice. This algorithm mimics RR, as requests 1,3, and 4 are granted equal processing time before preemption. However, unlike RR, it orders requests like SJF where the shortest request 2 is serviced first and the longest request 1 is serviced last.

RR-SJF allows all requests to get processing time within each round to improve fairness and TTFT, but prioritizes the requests which will finish earliest to improve E2EL. For instance, in the RR-SJF example in **Figure 1**, requests 2, 4, and 3 finish earlier than they do in RR (lower E2EL), while request 1 starts earlier than it does in SJF (lower TTFT). Time sharing in RR also has the secondary benefit of mitigating incorrect predictions in approximate SJF, ensuring that requests which are predicted to be shorter or longer than their actual lengths, and thus are not run in the correct order, will still receive sufficient processing time without completely blocking other requests.

Algorithm 1 Round-Robin Shortest Job First

Input: token and memory $budget$, $waiting$ queue, $running$ queue
Parameters: $steps\_before\_preemption$
Notes:
Priority: A value indicating the estimated number of output tokens the job will require in total. A lower value is higher priority.
Waiting Time: Time that a request has spent in the waiting queue since being placed on it.

if $waiting$ is empty **then**
  return
end if

Sort $waiting$ by waiting time with priority as tie break
Sort $running$ by priority

Select highest-priority request from $waiting$, R

Make list $eligible\_to\_preempt$ by adding all running requests (from $running$) that have produced at least $steps\_before\_preemption$ tokens

while $budget$ says memory resources are insufficient to schedule R and $eligible\_to\_preempt$ is not empty **do**
  Preempt a running request from $eligible_to_preempt$ and adjust resource allocations in $budget$
  Move preempted request from $running$ to $waiting$
end while

Schedule R to run if $budget$ indicates sufficient memory available

2.2 Implementation

The RR-SJF algorithm is implemented as a scheduling policy option in vLLM, a state-of-the-art LLM inference framework. Requests are submitted to vLLM with priority values representing their predicted output lengths. These length predictions can be generated by a separate fine-tuned BERT model instance.

vLLM batches requests to run on the GPU in each iteration, in which a token is generated for each request in the

decode stage. To simplify the implementation and improve performance, requests are only paused between iterations and the number of tokens generated is used as a proxy for processing time, i.e. requests are paused after a certain number of tokens have been generated instead of after a certain amount of time has elapsed. After each iteration, **Algorithm 1** preempts requests that have generated sufficient tokens and decides which requests to include in the next batch.

3 EVALUATION

We evaluate the performance of the RR-SJF scheduling algorithm in comparison to FCFS, SJF, and RR algorithms. we demonstrate the following:

1. RR-SJF can yield a lower TTFT than the baseline algorithms.
2. RR-SJF can yield a comparable E2EL to the optimal SJF scheduling algorithm.
3. RR-SJF is largely resistant to noisy output predictions, as the relative ordering of requests are still close to optimal.

3.1 Dataset Construction

We build a custom dataset from the Chatbot Arena dataset prompts (LMSYS, 2023), running each prompt through Llama 3.2-1B with 0.0 temperature to obtain a deterministic response. This gave us an "oracle" dataset of prompts where the exact number of output tokens are known for each prompt. Due to time constraints, we restrict the length of responses to 1024 tokens to shorten both dataset creation and metric evaluation time. We note that the majority of output response lengths matched the maximum 1024 output token length. To enhance variation in our request output lengths, we filter prompts sent to our model so that at least half of the prompts have fewer than 1024 output tokens.

3.2 Experimental Setup

We evaluate our scheduling algorithm with the Llama 3.2-1B model using an NVIDIA L4 GPU in Google Cloud. Note that we use recompute as the preemption mechanism, as swap preemption presented unforeseen challenges (see **Section 4.2** below).

All experiments are run in an online inference environment, where we serve Llama 3.2-1B on a local port through vLLM, then send a series of requests to the model via API call. As each request is sent individually, we can measure both TTFT and E2EL at the request granularity. We use the exact output lengths of the custom dataset as our "oracle" prediction to order requests in both our RR-SJF algorithm and the SJF baseline. We acknowledge that the majority of our metrics send a large number of requests almost simultaneously, which closely mimics an offline inference environment. We send many requests at once so that most of them will have to wait, allowing us to better measure TTFT and E2EL. However, we test in a more traditional online inference environment when exploring the impact of varied request rates below.

3.3 Tokens Generated Before Preemption

To effectively compare the RR-SJF algorithm with existing solutions, we must first determine the optimal number of tokens to generate before preemption so that we can minimize TTFT without sacrificing a reasonable E2EL. We test 5-200 tokens generated before preemption in increments of 5.

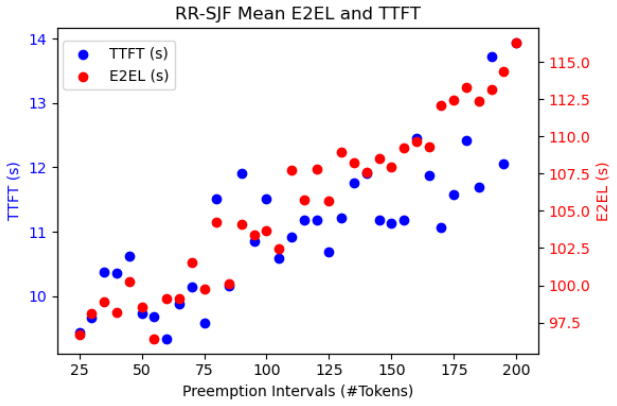

Figure 2. Double scatterplot of mean TTFT and E2EL over preemption intervals

From **Figure 2**, we observe a positive correlation between the number of tokens processed before preemption and the TTFT. This is because a lower number of tokens processed before preemption allows the algorithm to begin processing more requests much sooner, thus allowing a lower TTFT. We also observe a positive correlation between the number of tokens processed before preemption and E2EL. This result was surprising, as we expected latency to increase with an increased number of preemptions, which occurs with a lower number of tokens generated before preemption, indicating its overhead may be less significant than originally thought. The counterintuitive nature of the E2EL results cannot be accurately discerned at this time and require further investigation.

3.4 Scheduling Algorithm Comparison

We compare our novel RR-SJF scheduling algorithm against the following existing solutions: FCFS, SJF, and RR. For both RR-based algorithms, the number of tokens generated

is used as a proxy for processing time and we generate 5 tokens before preemption, as this minimizes TTFT without a significant increase in E2EL (see **Section 4.3**). For priority-based algorithms (SJF and RR-SJF), we use exact output lengths from our "oracle" dataset to determine priority.

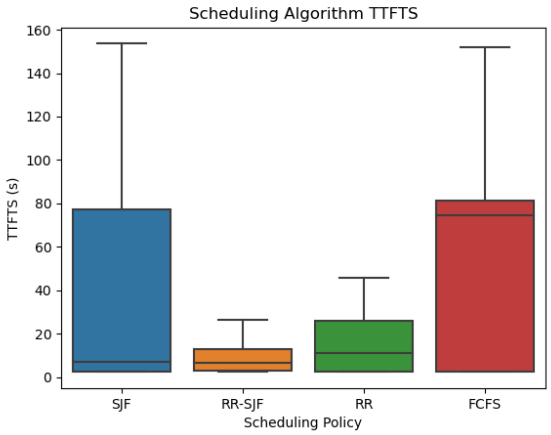

Figure 3. TTFTs - Preemption after generating 5 tokens

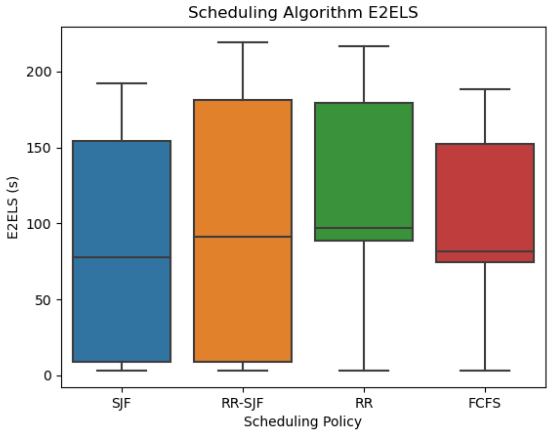

Figure 4. E2ELs - Preemption after generating 5 tokens

As expected, our RR-SJF scheduling algorithm yielded a lower median TTFT compared to the naive RR and FCFS scheduling algorithms (**Figure 3**). More specifically, the median TTFT in RR-SJF is 9x smaller than the median TTFT in FCFS because the smallest jobs are started early to reduce wait times. We see that SJF achieves a similar median TTFT as RR-SJF for the same reason.

We also observe that there is significantly less variation in TTFT with the RR-SJF algorithm compared to the baseline algorithm: RR-SJF's max TTFT is 5x smaller than SJF's and FCFS's and 1.5x smaller than RR's. This is by design, as RR-SJF consistently guarantees a lower TTFT by processing smaller requests first and periodically preempting requests to ensure longer requests can still start early.

The median end-to-end latencies between our custom algorithm and the baselines are similar (**Figure 4**). However, SJF and RR-SJF significantly improved the E2EL for the shortest requests by scheduling them to run first: the E2EL for RR-SJF is about 9x smaller for the 25th percentile than that of RR. RR-SJF does, however, have the largest max E2EL latency: the added time cost of preemption is borne the most by the longer output requests.

Pairing SJF with RR allowed our algorithm to perform better for shorter requests than a classic RR approach, while still yielding similar median and maximum E2ELs to an optimal SJF algorithm.

3.5 Adding Noise

In the above experiments, we first run offline inference of the model using a temperature of 0.0 to get the exact output lengths from the custom dataset. The above experimental runs show how the different scheduling algorithms act with the use of an "oracle" model, one that knows the exact outputs. This represents a perfect predictor model, which realistically will not happen. We wanted to use a BERT predictive model for this task, but had some challenges getting it working (see 4.1 for more discussion on this).

Due to this technical challenge, to simulate a real environment with an imperfect model, we instead introduce Gaussian-based noise into the dataset output length prediction. **Figures 5 and 6** show the TTFT and E2EL of the SJF and RR-SJF at different noise levels. We only chose these two policies because they are the only ones that depend on the predicted length. The random noise was based on a Gaussian distribution with $\mu = 0$ across different σ noises. As an example, for $\sigma = 25$ the predicted value given the request, oracle, and standard deviation for noise is formulated as $NewPrediction(req, \sigma = 25) = \mathcal{O}(req) + \mathcal{N}(0, \sigma^2)$. The new prediction is then given a hard floor of 1 output token and ceiling of 1024 tokens. This process is applied for each request independently. The TTFT graph (**Figure 5**) shows that the RR-SJF policy's TTFTs are not significantly affected by noise from the predictor model. On the other hand, SJF exhibits considerable variance in TTFT, with the 75th percentile TTFT changing dramatically. This indicates that the SJF algorithm is more prone to noise. The E2EL graph (**Figure 6**) shows that neither of these algorithms experience much variance for E2EL across different noise levels. One explanation for this is that adding Gaussian noise largely does not change the relative ordering of the jobs, only the predicted lengths. Even if the predicted lengths change, the shortest jobs will still be selected to

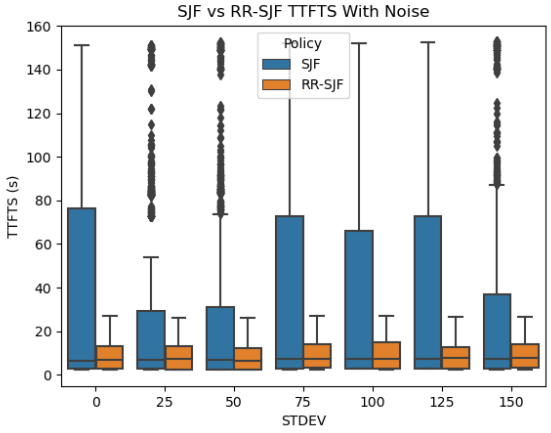

Figure 5. Boxplots comparing the TTFT across different noise levels of SJF and RR-SJF

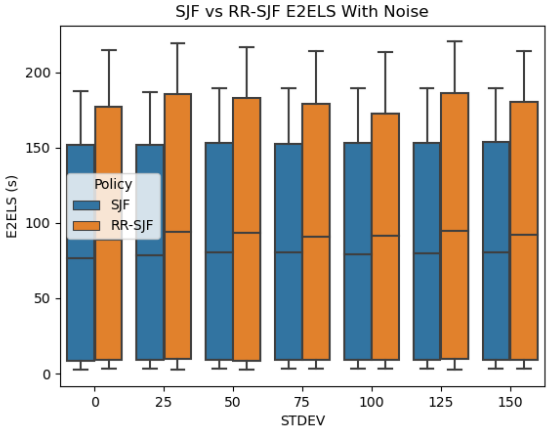

Figure 6. Boxplots comparing the E2EL across different noise levels of SJF and RR-SJF

be run first unless the lengths change enough to impact the relative ordering.

Further study should be done by inducing more aggressive noise (such as for models that misclassify long requests as small and vice versa) as well as for inaccurate predictive models.

3.6 Varying Request Rates

We acknowledge that requests are typically far more sporadic than the evaluation examined above. Therefore, we explore the effectiveness of our algorithm with varied request rates. We compare the RR-SJF algorithm to both SJF and FCFS algorithms via TTFT and E2EL comparisons. Again, we generate 5 tokens before preemption in RR-based algorithms, and prioritize requests with fewer output tokens.

Figure 7 and **Figure 8** reflect the same trend as in Section 3.4, where RR-SJF yields a lower TTFT at a slight cost of E2EL. Compared to FCFS, the median TTFT is reduced by nearly 10x when the request rate is at or above 25 req/sec. When compared to SJF, although RR-SJF has a higher median TTFT, the worst TTFT experienced by RR-SJF is approximately at the 60th percentile TTFT for SJF. This shows how RR-SJF is able to achieve good TTFT results, without extreme starvation, and underscores the consistency and fairness RR-SJF can provide.

Additionally, as request rate increases, we observe a more stable TTFT and E2EL. This stability arises because the server reaches its processing capacity, and sending more than 25 requests per second exceeds its ability to handle requests in real time, causing queuing.

4 Limitations

Here, we point out some of the limitations to the breadth and accuracy of the experiments we conducted for our scheduling method.

4.1 Use of Oracle

We were not originally planning to use an oracle. We were instead planning to (and did) fine-tune a regression-based BERT-110M model to predict sequence length which was similar to (Fu et al., 2024) that used an OPT-125M parameter model for a similar purpose. However, we were unable to get this model to even reasonably accurately predict output lengths.

Our main hypothesis for this is that the ShareGPT dataset we used ended up being a largely biased dataset (in the sense that generated responses were often greater than our cutoff of 1024 tokens). Since the optimization function was MSE, which heavily weights poor predictions, it biased the model to always predict high values. For our actual scheduling algorithm, we care less about the predicted value than the actual rankings of the predicted shortest jobs. However, the predictions were so off that the relative rankings weren't good either.

This motivated our decision to pivot and instead use an oracle and test different levels of Gaussian-generated noise to simulate an imperfect predictor model. One limitation of calculations being done offline is that the time it takes to predict the output token lengths is not considered in latency metrics.

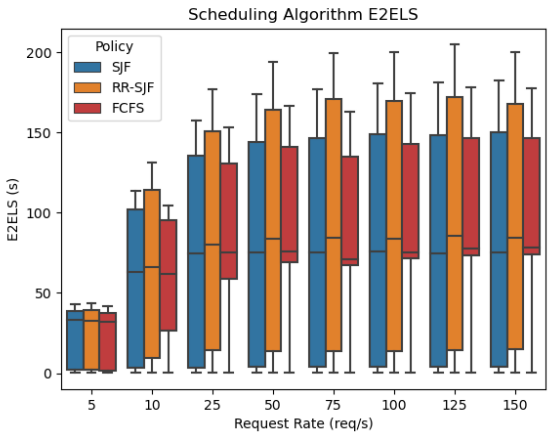

Figure 7. Boxplots comparing the E2EL across different request rates of SJF, RR-SJF, and FCFS

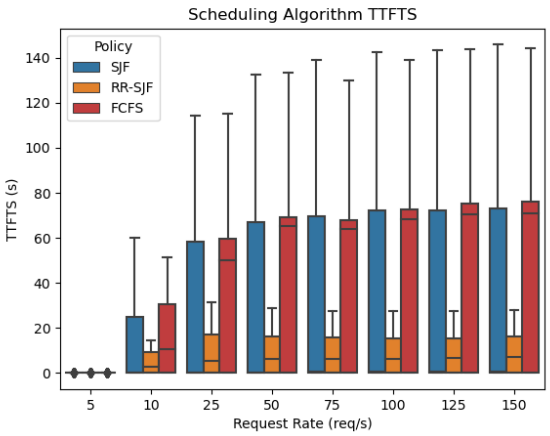

Figure 8. Boxplots comparing the TTFT across different request rates of SJF, RR-SJF, and FCFS

4.2 vLLM Swap vs Recompute

The RR-SJF scheduling strategy should work using both swap and recompute for KV cache memory management, but in our experiments, we were only able to test with recompute due to a bug in vLLM with preempting when in swap mode (vLLM's default priority preemption also fails due to the same bug). Although swap mode allows computed KV caches to be saved after a preemption, under certain circumstances, recompute may be faster due to avoiding the overhead of saving and reloading the cache.

With larger models, we expect that the time cost of recomputing the KV cache would be larger, thus it would have been useful to compare RR-SJF performance in both swap and recompute mode for inference with a very large model. Since the overhead associated with either mode scales for large models and long output lengths, RR-SJF may be less useful for very large models.

4.3 Dataset Homogeneity

To obtain an oracle dataset with deterministic, model-specific output lengths, we ran Chatbot Arena's prompts (LMSYS, 2023) through the Llama-3.2-1B model via vLLM's offline inference (Section 3.1).

To keep dataset creation and testing time feasible, we limited output lengths to a maximum of 1024 tokens. While this provided a decent range of output lengths (from 4 to 1024), we recognize that this isn't representative of actual output request lengths that Llama 3.2-1B may return, as it has a maximum concurrency of 8192 tokens. As a result, the restriction of output lengths in both our dataset and our evaluation may have prevented trends from manifesting, as our range of output lengths was relatively small.

Additionally, the majority of our prompts have an associated response token length of 1024, likely due to our small output restriction. To guarantee variation in our dataset, we filter 1000 random requests so at least half have output lengths with fewer than 1024 tokens. By forcing some variation into our dataset, we hoped to curate a more representative sample of output requests, even with our output restriction. However, we acknowledge that binning requests evenly based on whether or not the request returns the maximum output tokens may not be the most effective method of filtering prompts.

We observe in **Figure 9** that no matter the scheduling policy used, there are three humps in the E2EL distribution. This suggests that the data output lengths clustered around 3 values, rather than being uniformly distributed across the range of possible output lengths, as would have been preferred.

5 Related Work

LLM inference scheduling frameworks There are many inference scheduling speed-up strategies that focus on load distribution, request grouping, and memory management. Splitwise showed that scheduling can be improved by scheduling the prefill and decode stages of a request separately (Patel et al., 2024). There are also strategies to load balance incoming requests across GPU resources (Sun et al., 2024), choose optimal model, pipeline, and tensor parallelisms based on the request (Oh et al., 2024; Aminabadi et al., 2022), and kernel optimization (Aminabadi et al., 2022; Dao et al., 2022). Orca introduced request batching (Yu et al., 2022), and other works have shown that batches can be processed faster when formed based on prefix-sharing (Zheng et al., 2024a) and output length sim-

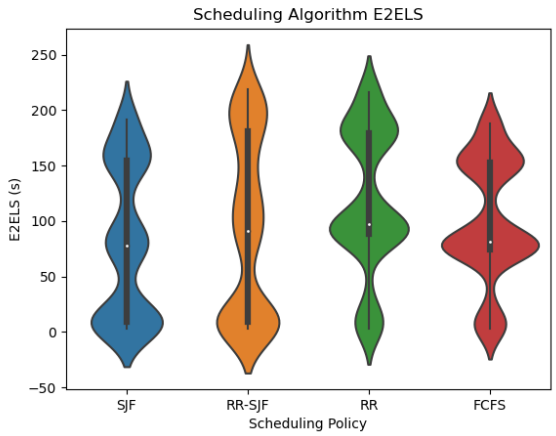

Figure 9. E2ELs - Violin plot of preemption after generating 5 tokens

ilarity (Zheng et al., 2024b; Guldogan et al., 2024). Many systems focus on optimal memory management, such as reducing memory fragmentation (Kwon et al., 2023; Sun et al., 2024; Wu et al., 2024a), or reducing KV cache memory movement (Stojkovic et al., 2024; Wu et al., 2024a).

While popular LLM serving frameworks such as vLLM (Kwon et al., 2023), TensorRT-LLM (NVIDIA, 2024a), Nvidia Triton (NVIDIA, 2024b), and the HuggingFace accelerate library (HuggingFace, 2024), have adopted many of these sophisticated strategies for managing the priority ordering jobs, most systems simply use First-Come First-Serve. The only time they reorder jobs with a preemptive policy is to respect user-defined priority to let high priority jobs run first. In practice, reordering request service priorities based on qualities of the request is not common despite possible advantages.

Preemptive scheduling algorithms

First-Come First-Serve, though popular, comes with large TTFTs and allows long requests at the front of the line to run to generate to completion uninterrupted even at the detriment of waiting requests. There are preemptive policies that attempt to mitigate these issues. The RR scheduling policy for LLMs can reduce average response time (Cheng et al., 2024). Andes developed a client-experience aware scheduling algorithm that dynamically adjusting job priorities based on user-perceived responsiveness while limiting preemption where possible due to its costliness, significantly reducing TTFT among other QoE metrics (Liu et al., 2024). Some works have developed speculative Shortest Job First scheduling (Fu et al., 2024; Qiu et al., 2024; Zheng et al., 2024b; Zhao & Wang, 2024) or shortest remaining time first scheduling (Shahout et al., 2024) by estimating the output length of requests. A well-known method from general queuing theory is least attained service scheduling, which favors shorter jobs without knowing job length in advance (Biersack et al., 2007). FastServe combines principles of least attained service and Shortest Job First by using a multi-level queue for priority of scheduling, where jobs with shorter prompts — acting as an heuristic for job length — are given higher priority at first, but then demoted to a lower priority queue if they don't finish after some time allotment, allowing less served prompts to be scheduled (Wu et al., 2024b).

Output length prediction

Speculative Shortest Job First scheduling requires a job length prediction method. FastServe uses prompt length as a heuristic for total job length: prompt length is great for predicting the time spent in the prefill stage, though its accuracy for output length prediction is limited (Wu et al., 2024b). It's been shown that a small proxy model, such as a fine-tuned BERT-base model or other fine-tuned LLM, when given the prompt, can predict the length of the output that will be produced by a different model (Qiu et al., 2024; Zheng et al., 2024b; Abgaryan et al., 2024; Hu et al., 2024) or determine the relative rankings of output lengths for a set of prompts (Fu et al., 2024). TRAIL uses an embedding of the output-producing LLM's internal structure to predict remaining output length, making do with just one model (Shahout et al., 2024). ALISE uses embeddings to predict output length: an embedding is made of the request's prompt and then use a query database and an MLP decoder are used to decode the output length (Zhao & Wang, 2024).

6 Future Work

This paper has assumed that Shortest Job First perfectly minimizes latency, but that may not always be true such as if throughput in LLM inference is limited by GPU memory. Requests with very large input sizes, and thus large KV caches, may consume enough memory that they prevent some requests from being included in batches, decreasing throughput and increasing latency. In such a case, prioritizing the shortest jobs may not always minimize latency and batching optimally also needs to be considered. Future work could seek to find an algorithm which optimizes these cases and then apply them to RR similar to RR-SJF.

The implementations of RR and RR-SJF used in this paper treat the number of tokens generated as a proxy for processing time to improve throughput and simplify the implementations. This does not account for time spent computing or swapping KV caches, differences in scheduling overhead between batches (smaller batches may use less processing time), or differences in time to generate tokens in the same request (previously generated tokens are used

as inputs for later tokens). These factors likely make predicting total latency more difficult than predicting output length. The benefits and drawbacks of using the number of generated tokens as a proxy could be discovered by future work.

RR-SJF keeps time slice durations (tokens generated each time slice) constant across all requests, just like in RR, and this paper in general does not consider adjusting time slice durations for different requests. Varying time slice duration for each request based on predicted request output length, e.g. making each request's time slice equivalent to half the time required to generate the predicted length, could allow finer-grain tradeoffs between fairness and latency. Another case in which this could be useful is allocating more processing time for requests which remain outstanding for an excessively long time to improve fairness (depending on the definition of fairness used). Testing the benefits of varying time slice durations is left for future work.

7 Conclusion

In this paper, we present Round-Robin Shortest Job First scheduling for inference serving. Our algorithm improves median TTFT by around 9x compared to FCFS by scheduling the smallest jobs first and reducing head-of-line blocking, while maintaining similar, though slightly larger, median E2EL. RR-SJF has advantages over both RR alone and SJF alone. While RR-SJF has a similar median TTFT to SJF, the max TTFT is around 5x smaller showing RR-SJF begins serving long requests sooner. RR-SJF has a similar median E2EL to RR, but the 25th percentile of E2ELs is around 9x smaller because small requests are allowed to run first. In summary, Round-Robin Shortest Job First scheduling demonstrates a balanced and effective approach for inference serving, significantly reducing TTFT and improving overall responsiveness for shorter requests, while maintaining fairness and competitive E2EL performance.

References

Abgaryan, H., Harutyunyan, A., and Cazenave, T. Llms can schedule. *arXiv preprint arXiv:2408.06993*, 2024.

Aminabadi, R. Y., Rajbhandari, S., Zhang, M., Awan, A. A., Li, C., Li, D., Zheng, E., Rasley, J., Smith, S., Ruwase, O., and He, Y. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022. URL https://arxiv.org/abs/2207.00032.

Biersack, E. W., Schroeder, B., and Urvoy-Keller, G. Scheduling in practice. *SIGMETRICS Perform. Eval. Rev.*, 34(4):21–28, March 2007. ISSN 0163-5999. doi: 10.1145/1243401.1243407. URL https://doi.org/10.1145/1243401.1243407.

Cheng, K., Hu, W., Wang, Z., Peng, H., Li, J., and Zhang, S. Slice-level scheduling for high throughput and load balanced llm serving, 2024. URL https://arxiv.org/abs/2406.13511.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Fu, Y., Zhu, S., Su, R., Qiao, A., Stoica, I., and Zhang, H. Efficient llm scheduling by learning to rank. *arXiv preprint arXiv:2408.15792*, 2024.

Guldogan, O., Kunde, J., Lee, K., and Pedarsani, R. Multi-bin batching for increasing llm inference throughput, 2024. URL https://arxiv.org/abs/2412.04504.

Hu, C., Huang, H., Xu, L., Chen, X., Xu, J., Chen, S., Feng, H., Wang, C., Wang, S., Bao, Y., Sun, N., and Shan, Y. Inference without interference: Disaggregate llm inference for mixed downstream workloads, 2024. URL https://arxiv.org/abs/2401.11181.

HuggingFace. Accelerate: A library for simple and distributed training. https://github.com/huggingface/accelerate, 2024. Accessed: 2024-12-13.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Liu, J., Wu, Z., Chung, J.-W., Lai, F., Lee, M., and Chowdhury, M. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services, 2024. URL https://arxiv.org/abs/2404.16283.

LMSYS. Chatbot arena conversations dataset. https://huggingface.co/datasets/lmsys/chatbot_arena_conversations, 2023. Accessed: 2024-12-16.

NVIDIA. Tensorrt. https://github.com/NVIDIA/TensorRT-LLM, 2024a. Accessed: 2024-12-13.

NVIDIA. Triton inference server. https://github.com/triton-inference-server/server, 2024b. Accessed: 2024-12-13.

Oh, H., Kim, K., Kim, J., Kim, S., Lee, J., seong Chang, D., and Seo, J. Exegpt: Constraint-aware resource scheduling for llm inference, 2024. URL https://arxiv.org/abs/2404.07947.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, , Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019.

Qiu, H., Mao, W., Patke, A., Cui, S., Jha, S., Wang, C., Franke, H., Kalbarczyk, Z. T., Başar, T., and Iyer, R. K. Efficient interactive llm serving with proxy model-based sequence length prediction, 2024. URL https://arxiv.org/abs/2404.08509.

Shahout, R., Malach, E., Liu, C., Jiang, W., Yu, M., and Mitzenmacher, M. Don't stop me now: Embedding based scheduling for llms, 2024. URL https://arxiv.org/abs/2410.01035.

Stojkovic, J., Zhang, C., Íñigo Goiri, Torrellas, J., and Choukse, E. Dynamollm: Designing llm inference clusters for performance and energy efficiency, 2024. URL https://arxiv.org/abs/2408.00741.

Sun, B., Huang, Z., Zhao, H., Xiao, W., Zhang, X., Li, Y., and Lin, W. Llumnix: Dynamic scheduling for large language model serving, 2024. URL https://arxiv.org/abs/2406.03243.

Wu, B., Liu, S., Zhong, Y., Sun, P., Liu, X., and Jin, X. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism, 2024a. URL https://arxiv.org/abs/2404.09526.

Wu, B., Zhong, Y., Zhang, Z., Liu, S., Liu, F., Sun, Y., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models, 2024b. URL https://arxiv.org/abs/2305.05920.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Zhao, Y. and Wang, J. Alise: Accelerating large language model serving with speculative scheduling, 2024. URL https://arxiv.org/abs/2410.23537.

Zheng, Z., Ji, X., Fang, T., Zhou, F., Liu, C., and Peng, G. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching, 2024a. URL https://arxiv.org/abs/2412.03594.

Zheng, Z., Ren, X., Xue, F., Luo, Y., Jiang, X., and You, Y. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36, 2024b.