ALISE: Accelerating Large Language Model Serving with Speculative Scheduling

Youpeng Zhao, Jun Wang

Computer Systems and Data Science (CASS) Lab,

University of Central Florida

Background

UCF

Large Language Models (LLMs)

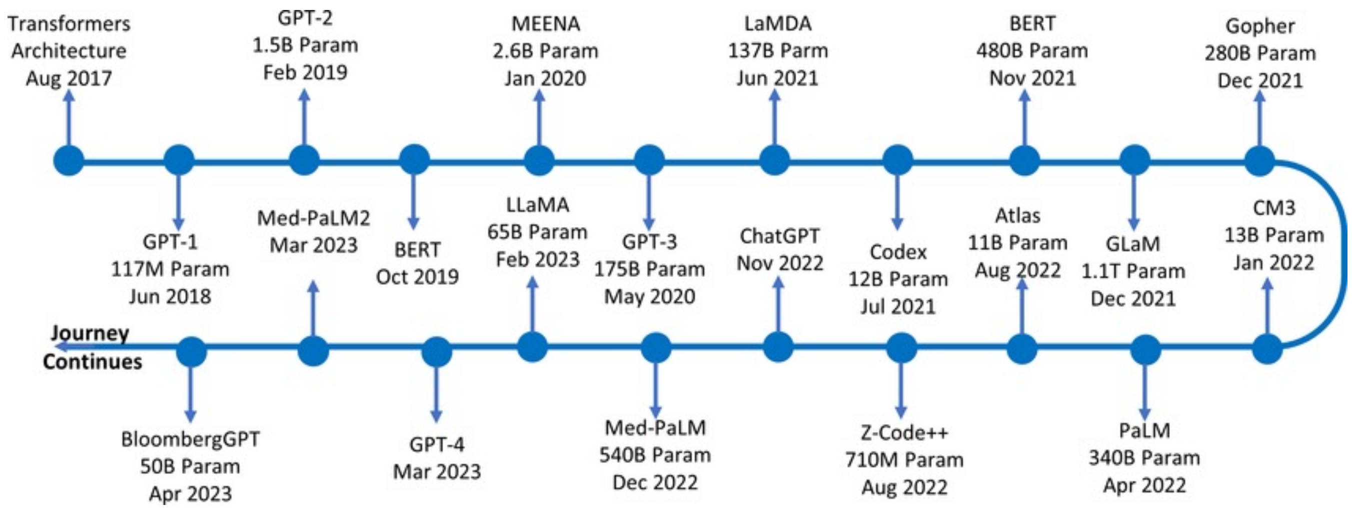


Fig. 1: A timeline of large language model and its applications in recent years.

ChatGPT

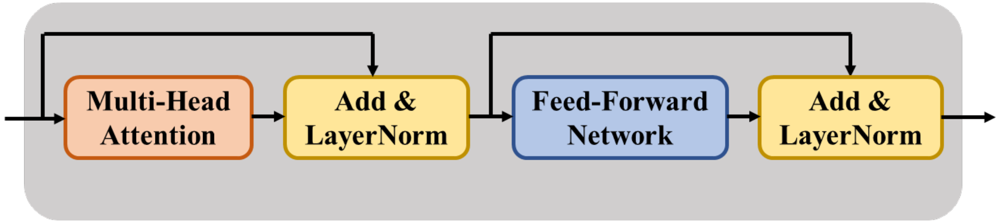
Gemini

Claude

Inference serving accounts for most LLM-based application scenarios. Accelerating LLM serving has become an increasingly important research problem.

Background

Characteristic of LLM Inference



$$AW(Q,K) = \sigma\left(\frac{QK^T}{\sqrt{d}}\right)$$

$$Attn(Q,K,V) = AW(Q,K) \cdot V$$

1. **Multi-iteration**: require multiple passes of the model
2. **Auto-regressive**: future tokens depend on previous tokens
3. **KV Cache**: need to store intermediate key-value states

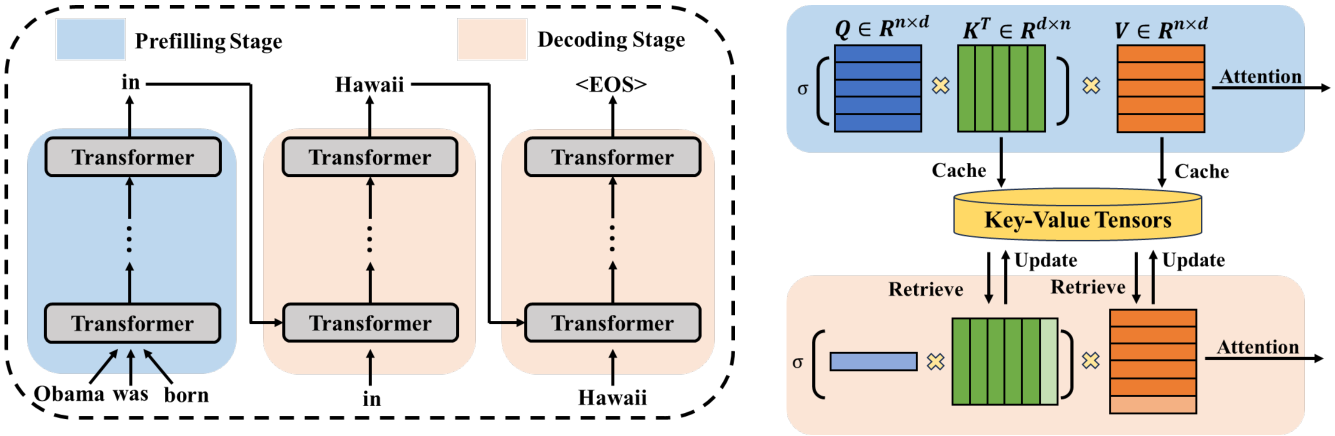


Fig. 2: Left: Autoregressive inference of LLMs; Right: KV Cache Mechanism.

Motivation

Multi-tenant Online Serving Systems

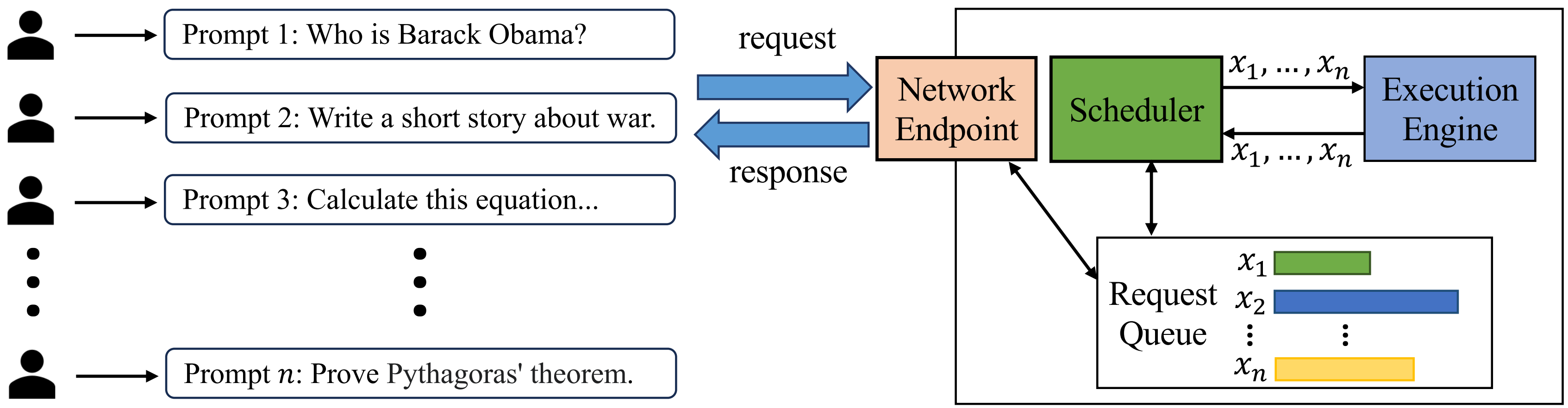


Fig. 3: An example of LLM serving system.

LLM Inference Systems

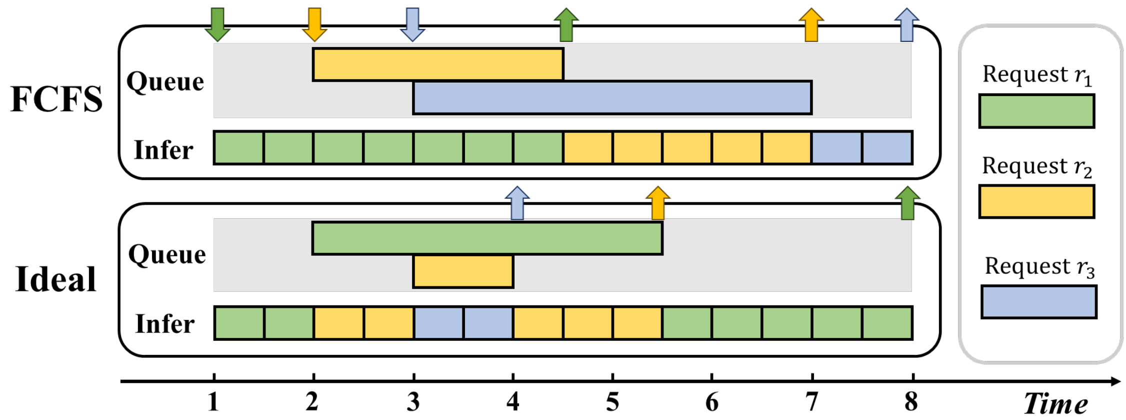
- **FasterTransformer (NVIDIA)**: custom CUDA kernels
- **ORCA (OSDI 22):** continuous batching and iteration-level scheduling
- **vLLM (SOSP 23):** non-contiguous paged memory to store KV cache at block-level

However, all these systems employ the first-come-first-serving (FCFS) policy, which could lead to potential head-of-line (HoL) blocking issues.

Main Problem

Head-of-Line (HoL) Blocking

- HoL is relatively mild when the incoming requests are homogeneous (same sequence length)
- HoL becomes prominent when the requests are heterogeneous and exhibits high variances, which can cause high response latency, as late arrived short jobs must wait for the previous long jobs to finish



Avg. Response Latency:

$$T_{FCFS} = \frac{(4.5-1)+(7-2)+(8-3)}{3} = 4.5$$

$$T_{Ideal} = \frac{(8-1)+(5.5-2)+(4-3)}{3} = 3.8$$

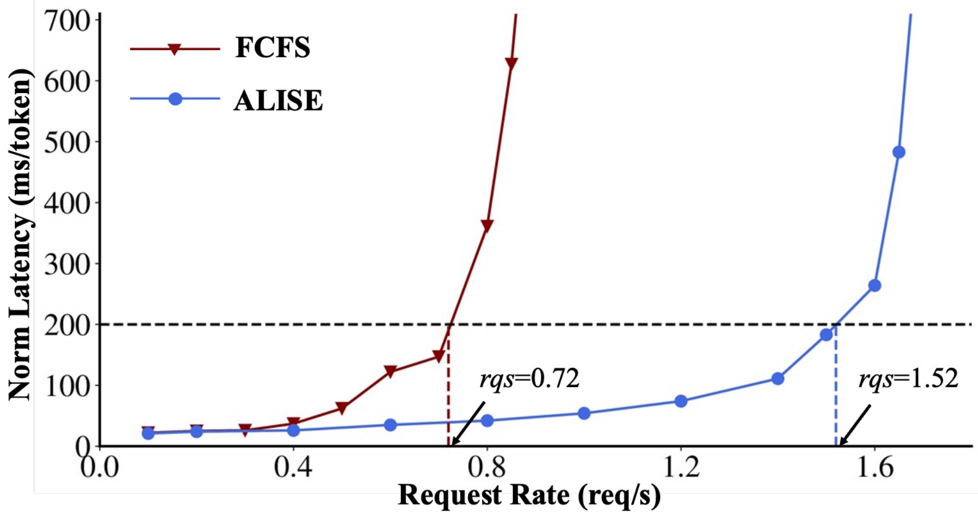


Fig. 4: Performance comparison of FCFS and ALISE.

Proposal

Is it possible to enable preemptive scheduling for LLM inference serving?

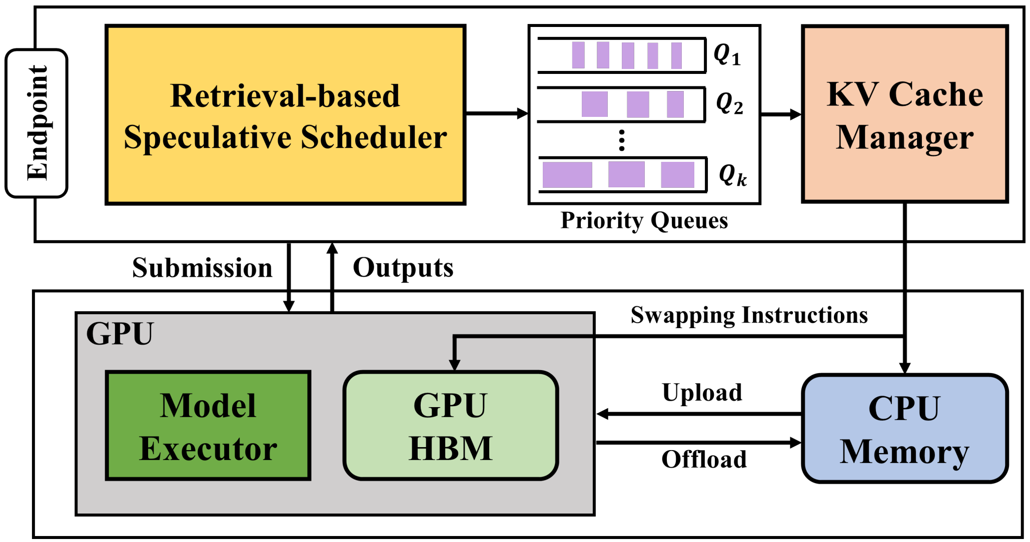


Fig. 5: ALISE System Overview.

1. If we can estimate (speculate) the execution time of heterogenous LLM requests, we could assign appropriate job priority orders to minimize response latency
   - **Retrieval-based Speculation (length prediction)**
   - **Mathematical Modeling (execution prediction)**
   - **Priority Queues (implementation)**

2. Preempted jobs have corresponding intermediate states (KV cache), we need efficient memory management.
   - **Priority-based Dynamic Swapping (reduce I/O latency)**
   - **KV Compression (reduce memory overhead)**

Scheduler Design

1. **Retrieval-based Speculative Model**: predict the output sequence length
   - Pre-process the input prompt with a BERT encoder
   - Perform length prediction using an ensemble of vector database (DB) and MLP decoder
2. **Mathematical Modeling**: estimate the execution runtime
   - The execution can be divided into two stages, prefilling and decoding

$$T_{gen} = T_{pre}(s) + T_{dec}(s,n)$$

$$T_{pre} \approx s \cdot T_0, T_{dec} \approx n \cdot (\alpha s + \beta)$$

$$T_{gen} \approx s \cdot T_0 + n \cdot (\alpha s + \beta)$$

   - $\{\alpha, \beta, T_0\}$ can be determined via benchmark profiling beforehand

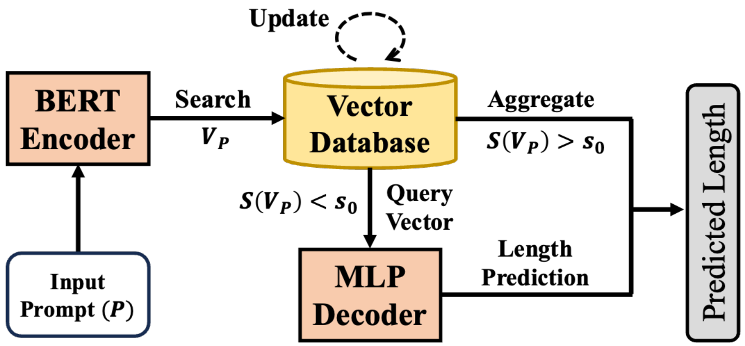


Fig. 6: Retrieval-based Length Predictor Architecture.

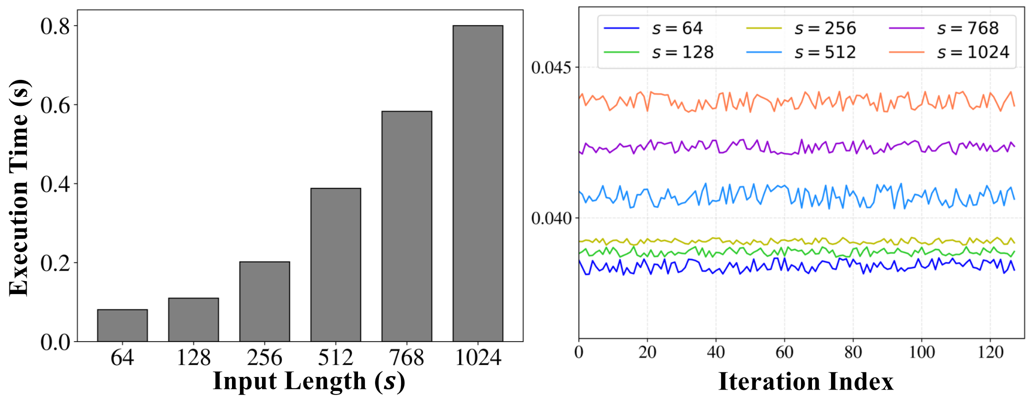


Fig. 7: Execution time comparison for prefilling (left) and decoding (right) for different input and output lengths.

Memory Management

Strawman solutions:

1. **Defer**: defer new arrived jobs when GPU memory reaches its capacity

   Jobs with higher priority (shorter execution time) are blocked (HoL issues)

2. **Recompute**: delete the intermediate states of preempted jobs, and recompute them when needed

   Recomputation induces additional latency; potential deadlocks in scheduling

Dynamic Swapping:

- Dynamically offload and upload the intermediate states
- We perform swapping operations based on the estimated wait time (EWT), which considers both the priority orders and aging mechanism
- Jobs with high EWT will be offloaded to CPU memory, and jobs with low EWT will be uploaded for execution later

KV Compression:

- Fine-grained channel-wise quantization for intermediate KV cache
- Quantization from FP16 to INT8 (reduce memory consumption by half)

Algorithm 2 Dynamic Swapping

Input: Priority job queue $\{Q_i\}_{i=1,\dots,h}$, GPU memory GM, CPU memory CM, GPU job limit M.

while true **do**
    for $q \in \{Q_1, \dots, Q_h\}$ **do**
        EWT(q).$sort()$
        for $i = 1$ **to** $len(q)$ **do**
            if J_i not in GPU **and** $i < M$ **then**
                # Preemptive upload
                CM.upload(J_i)
            else
                # Preemptive offload
                GM.offload(J_i)
            end if
            # Update job limit by each queue
            if $len(q) < M$ **then**
                $M = M - len(q)$
            end if
        end for
    end for
end while

Evaluation

Experimental Settings:

- Models: OPT (2.7B, 6.7B, 13B), LLaMA (7B, 13B), Pythia (12B)
- Baselines: ORCA (OSDI 22), vLLM (SOSP 23), Oracle (upper bound)
- Datasets: Alpaca, ShareGPT
- Metrics: Normalized latency
- Hardware: 2 x Tesla V100 32 GB, 1024 GB DRAM
- Implementation: vLLM (PyTorch), 4K lines of Python and 1.5K lines of C++.

Main Results

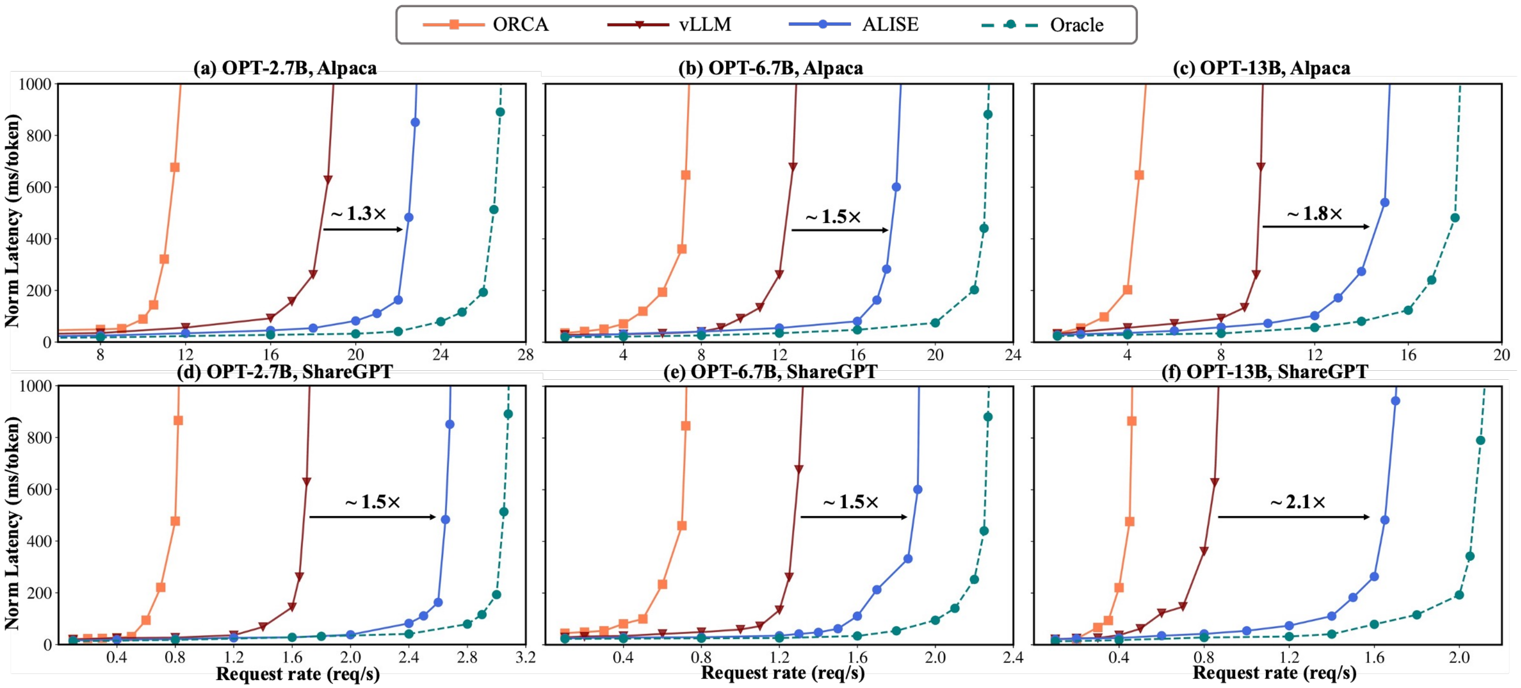


Fig. 8: End-to-end performance of ALISE, vLLM, ORCA and Oracle.

Observations:

- ALISE achieves 1.3~1.8 x improvement under same latency constraints on the Alpaca dataset
- On the ShareGPT dataset with higher length variations, ALISE sustains up to 2.1 x improvement against vLLM

Ablation Study

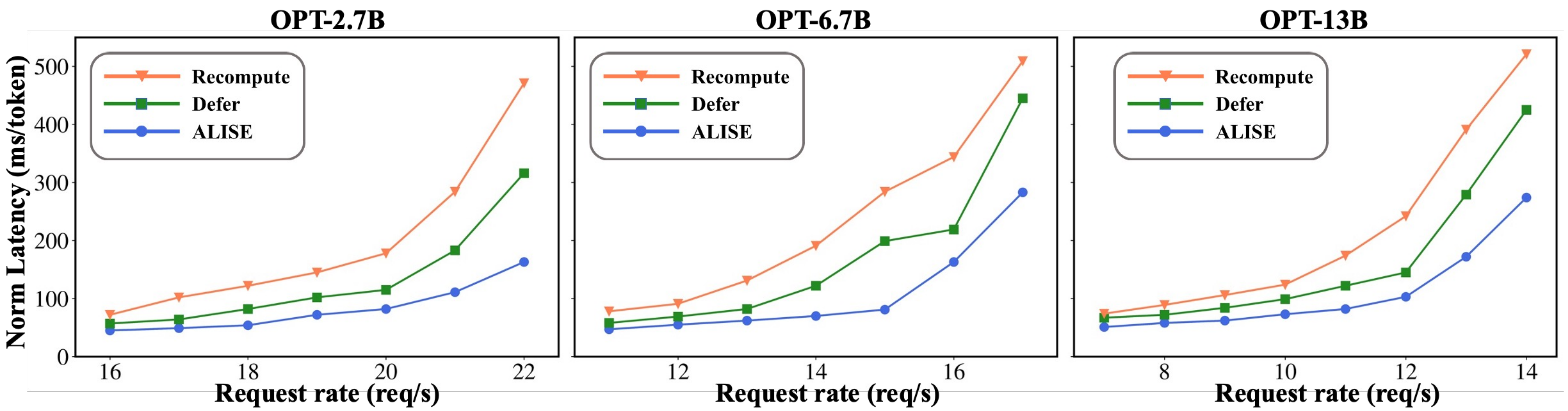


Fig. 9: Latency comparison of different memory management mechanisms in ALISE

Table 1: Throughput improvement for different LLM models.

	ORCA	vLLM	ALISE
Alpaca (30 req/s)			
LLaMA-13B	42.23	71.87	**101.42** (+41%)
LLaMA-7B	75.42	122.87	**164.51** (+34%)
Pythia-12B	34.85	64.19	**91.12** (+42%)
ShareGPT (2 req/s)			
LLaMA-13B	14.42	27.89	**41.22** (+47%)
LLaMA-7B	30.28	62.93	**87.32** (+39%)
Pythia-12B	12.36	24.95	**37.27** (+49%)

Table 2: Accuracy and throughput improvement for proxy-based and our method.

Metrics	OPT-2.7B		OPT-6.7B		OPT-13B	
	Proxy-based	Retrieval-based	Proxy-based	Retrieval-based	Proxy-based	Retrieval-based
Accuracy (↑)	0.781	**0.821**	0.712	**0.856**	0.634	**0.744**
Pred. Error (↓)	0.122	**0.057**	0.145	**0.096**	0.178	**0.123**
Avg. Pred. Latency (↓)	12.2 ms	**3.92 ms**	11.7 ms	**4.74 ms**	14.8 ms	**4.49 ms**
Throughput (↑)	1×	1.47×	1×	1.63×	1×	1.82×

Observations:

- Our dynamical swapping consistently outperforms strawman solutions
- Our retrieval-based predictor can provide much better accuracy with lower overhead
- ALISE can also generalize across different LLM models

Conclusion

- We identify the scheduling bottlenecks, i.e., the head-of-line (HoL) blocking issues in existing FCFS policy for LLM inference serving

- We propose a speculative scheduler that leverages a retrieval-based predictor to estimate the execution time of each incoming job and priority queues that preemptively schedules workloads to minimize the response latency

- To alleviate the memory management issues, we design an adaptive memory manager that dynamically performs preemptive offload/upload operations for unused intermediate states and compress KV cache with quantization

- Based on these two techniques, we present ALISE, a system prototype to accelerate LLM inference serving. Experiments demonstrate that ALISE obtains significant throughput improvement over the state-of-the-art systems, such as vLLM and ORCA

Acknowledgements

- We appreciate the generous support from the National Science Foundation (NSF) grant 1907765, 2400014

- Faculty and student investigators from the UCF Computer Systems and Data Science (CASS) Laboratory

Thank you!