ALISE: Accelerating Large Language Model Serving with Speculative Scheduling

Youpeng Zhao
University of Central Florida
Orlando, FL, USA
youpeng.zhao@ucf.edu

Jun Wang
University of Central Florida
Orlando, FL, USA
jun.wang@ucf.edu

ABSTRACT

Large Language Models (LLMs) represent a revolutionary advancement in the contemporary landscape of artificial general intelligence (AGI). As exemplified by ChatGPT, LLM-based applications necessitate minimal response latency and maximal throughput for inference serving. However, due to the unpredictability of LLM execution, the first-come-first-serve (FCFS) scheduling policy employed by current LLM serving systems suffers from head-of-line (HoL) blocking issues and long job response times.

In this paper, we propose a new efficient LLM inference serving framework, named ALISE. The key design paradigm of ALISE is to leverage a novel speculative scheduler by estimating the execution time for each job and exploiting such prior knowledge to assign appropriate job priority orders, thus minimizing potential queuing delays for heterogeneous workloads. Furthermore, to mitigate the memory overhead of the intermediate key-value (KV) cache, we employ a priority-based adaptive memory management protocol and quantization-based compression techniques. Evaluations demonstrate that in comparison to the state-of-the-art solution vLLM, ALISE improves the throughput of inference serving by up to 1.8× and 2.1× under the same latency constraint on the Alpaca and ShareGPT datasets, respectively.

ACM Reference Format:
Youpeng Zhao and Jun Wang. 2024. ALISE: Accelerating Large Language Model Serving with Speculative Scheduling. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676659

1 INTRODUCTION

In recent years, large language models (LLMs) have demonstrated ubiquitous performance across various natural language processing (NLP) tasks [1, 3, 30, 37]. Notably, LLMs can generalize well to perform multiple tasks without additional fine-tuning, which makes them versatile and adaptable for downstream AI applications [17]. Inference serving is crucial to applications driven by LLMs, such as AI chatbots. The interactive nature of such applications requires the system to produce inference results for user requests at low latency and high throughput.

ACM ISBN 978-1-4503-XXXX-X/24/10
https://doi.org/10.1145/3676536.3676659

The inference process of LLMs exhibits the unique characteristics of *autoregressiveness*, where LLMs generate new tokens sequentially based on the input (prompt) tokens and all prior generated tokens. LLM inference typically requires running multiple iterations of the model to complete the token generation. General DNN inference serving systems [5, 10, 23, 36] focus on deterministic workloads such as image classification and object detection, which are highly predictable. The autoregressive nature makes these solutions not work and scale well for LLM workloads, as in LLM inference, each token is generated based on the preceding ones, and this sequential dependency can result in unpredictable execution runtime for generated sequences.

Existing LLM systems focus on improving serving throughput performance with custom CUDA kernels [6, 7, 16], iteration-level scheduling [34, 35], and efficient memory management [13, 25, 39]. Unfortunately, despite large strides toward improving the performance of serving LLMs, today's system platforms continue to struggle to provide low latency, especially in real-world workloads. We argue that this struggle stems from the non-preemptive first-come-first-served (FCFS) [38] scheduling strategy, which causes head-of-line (HoL) blocking issues when processing heterogeneous workloads. Run-to-completion scheduling is recognized for its HoL blocking issue, which is especially troublesome for generative inference tasks. This occurs when prior scheduled tasks, potentially lengthy in execution, hinder the timely processing of subsequent shorter tasks. Such queuing delays could significantly deteriorate the quality of service (QoS) provided. How to efficiently handle heterogeneous requests with variable execution time and perform appropriate scheduling to maintain high throughput remains a challenging problem.

In this paper, we introduce ALISE, to accelerate LLM serving with a novel speculative scheduler. A key insight behind ALISE is to estimate the execution time of heterogeneous LLM requests. Specifically, we adopt a retrieval-based length predictor, where a user prompt is first embedded into high-dimension vectors using a pre-trained text encoder, and the output length is accurately predicted using an ensemble of a query database and an all-MLP decoder. Based on such prior knowledge, we can estimate each job's execution time and leverage priority queues to perform preemptive scheduling, thus alleviating HoL issues and minimizing the total response time. To efficiently manage the intermediate KV cache of preempted jobs, we design a priority-based adaptive memory management mechanism that dynamically performs swapping operations (e.g., upload and offload) of the KV cache for preempted jobs based on each job's estimated wait time (EWT). Furthermore, we utilize the quantization technique to compress the KV cache to lower precision further reducing the overall memory overhead.

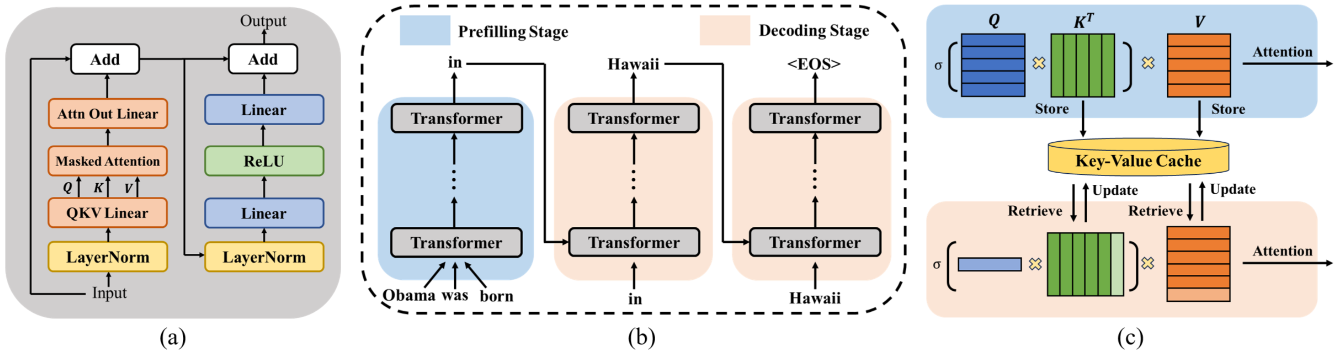

Figure 1: (a) Operations in the transformer layer. (b) An illustrative example of the autoregressive LLM inference process. (c) KV cache mechanism: at the prefilling stage, all input tokens are processed simultaneously, and the KV cache is initialized; at the decoding stage, the stored KV cache is retrieved for reuse and updated by iteration until termination.

We implement and evaluate ALISE across different serving scenarios. Extensive evaluations demonstrate that ALISE outperforms the existing FCFS solutions in end-to-end latency and overall throughput. Specifically, ALISE sustains up to 1.8× and 2.1× throughput improvement against the state-of-the-art solution, vLLM [13], on the Alpaca [27] and ShareGPT [28] datasets, respectively.

In summary, we make the following contributions:

- We propose a novel retrieval-based speculative scheduler to estimate the execution time of each incoming job and schedule workloads based on their priorities to minimize the response latency.
- We design an adaptive memory manager that intelligently performs preemptive offload/upload operations for unused intermediate KV cache and quantization to reduce memory overhead.
- We evaluate our serving solution, termed ALISE, on different scenarios and show that it significantly outperforms the existing LLM inference solutions, such as ORCA [35] and vLLM [13].

The remainder of this paper is organized as follows. Section 2 presents the background and motivation for our work. Then, Section 3 elaborates on the methodology of our system design, with evaluation followed in Section 4. Finally, Section 5 recaps related work, and Section 6 concludes this work.

2 BACKGROUND AND MOTIVATION

Transformer Architecture. The Transformer architecture [31] has significantly advanced natural language processing (NLP) and has been foundational in large language models (LLMs). A standard transformer model comprises an embedding layer to project sequences of tokens to hidden dimensions and stacks of transformer layers to capture long-term dependencies between input tokens using the self-attention mechanism. The embedding layer is responsible for converting discrete words (tokens) into high-dimensional vectors that can be processed by neural networks, capturing both the semantic and positional information of individual tokens [15]. A transformer layer includes two main components: a multi-head attention (MHA) module and a position-wise feed-forward network (FFN), as shown in Figure 1(a). The MHA module facilitates capturing contextual information by attending to different positions within the input sequence, while the FFN performs element-wise transformations to introduce non-linearity and improve representational capacity.

At the core of transformer layers lies the attention module [31]. The relevant operations are given in Equation 1 and 2. The LayerNorm and residual connection are omitted for simplicity. Three intermediate tensors are involved, namely, query Q, key K, and value V. The attention weights are calculated by first computing the dot product between Q and K, then scaling the product by the square root of hidden dimension d divided by the number of heads h, and finally going through a softmax operation ($\sigma(\cdot)$). The attention scores $\text{Attn}(Q, K, V)$ are calculated by multiplying the attention weights by V.

$$\text{Attn}(Q, K, V) = \sigma(\frac{QK^T}{\sqrt{d/h}}) \cdot V \tag{1}$$

The MHA output is obtained by simply concatenating the outputs of all attention heads along the head dimension, with each head being an attention module.

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{Attn}_1, ..., \text{Attn}_h) \tag{2}$$

LLM Inference. The generative inference of LLMs adopts an incremental decoding approach where the system computes the activations for all input prompt tokens in a single step and then iteratively decodes one new token using the input prompt and all previously generated tokens. Figure 1(b) gives an example of such autoregressive behavior in the LLM inference. The inference procedure of the LLM inference can be divided into two parts, including the prefilling and decoding stages. As shown in the example, during the prefilling stage, LLMs first process all the input tokens in a single pass; then, during the decoding stage, a previously generated output token is fed back into the model as an input and generates the next output token. Therefore, the decoding stage unfolds iteratively, processing one token at a time, until the sequence length reaches a maximum threshold or an "⟨EOS⟩" token is emitted.

The distinctive autoregressive characteristics open up the opportunity of reusing intermediate states, specifically, the key (K)

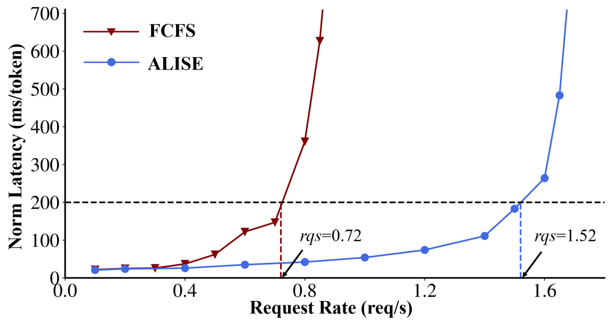

Figure 2: End-to-end performance comparison of existing FCFS scheduling and speculative scheduling in ALISE on the ShareGPT dataset.

and value (V) tensors in the transformer layers, often referred to *KV cache* [18, 21]. Note that the KV cache of one token depends on all previous tokens, and the size of the KV cache increases linearly as sequence length grows, as shown in Figure 1(c). With the KV cache, the original quadratic-complexity calculation is replaced with linear-complexity memory accesses, thus significantly speeding up the LLM inference at the expense of additional GPU memory overhead.

HoL Blocking in today's LLM Serving Systems. In LLM inference, each token is generated based on the preceding ones, and this sequential dependency can result in unpredictable output lengths for generated sequences. The execution time for each request exhibits significant variability among input prompts, depending on both the input prompt and output length (number of iterations). In FCFS, once a workload is scheduled, it must run until it finishes. Run-to-completion scheduling is recognized for its head-of-line blocking issue, which is especially troublesome for generative inference tasks. This occurs when prior scheduled tasks, potentially lengthy in execution, hinder the timely processing of subsequent shorter tasks.

To showcase such HoL blocking issues, we run the OPT-13B [37] model on a single NVIDIA V100 GPU on the ShareGPT dataset [28] with varying sequence lengths to benchmark the baseline FCFS and ALISE. Figure 2 shows the end-to-end latency comparison of both methods. As the request rate continually increases, FCFS induces much higher response latency due to inflexible non-preemptive scheduling. In contrast, ALISE, with accurate and timely information about sequence length and execution time, can schedule the workloads appropriately, maximizing device usage, and providing up to 2× better throughput under the same latency constraints.

3 METHODOLOGY

In this work, we develop a new speculative scheduling mechanism for LLM inference and build ALISE to tackle the challenges outlined in Section 2. Figure 3 illustrates the architecture of ALISE. When users submit their requests to the inference service, the retrieval-based speculative scheduler first predicts the executive runtime and sends the job profile to priority queues to assign the appropriate priority level at the granularity of iteration, which favors the job with the shortest remaining time to address the HoL blocking issues.

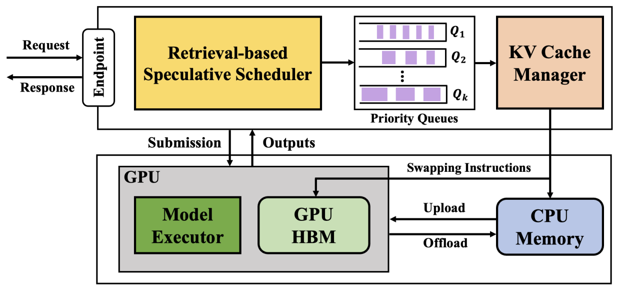

Figure 3: System overview of ALISE.

Once a job is selected, it will be automatically submitted to the GPU model executor, which retrieves the corresponding KV cache for computation. ALISE also features an adaptive KV Cache manager that dynamically swaps the KV cache of preempted jobs between GPU HBM and CPU memory that overlaps with computation to avoid additional latency.

3.1 Speculative Scheduler

Output Length Prediction. A key aspect of enabling preemptive scheduling for LLM workloads is to predict the output sequence length. Previous works have attempted to perform sequence prediction using smaller LLMs [19, 40] or proxy models [12, 22], however, they generally impose extensive engineering efforts and incur non-negligible overhead. In this work, we propose to adopt a retrieval-based approach that leverages a vector database (DB) for estimating output length with high accuracy and low memory and latency overhead. Figure 4 illustrates the details of our method.

We first transform the user prompt into high-dimension vectors using a pre-trained BERT encoder network [8]. Next, we conduct a similarity search for input in the vector DB and retrieve top-k historical queries with similarity scores above a pre-defined threshold. The final output length is predicted using the weighted average of retrieved queries. If no similar queries are found in the vector

Algorithm 1 Retrieval-based Length Prediction

```
Input: Input prompts $\{P_j\}_{j=1,\dots,n}$, similarity threshold $s_0$, query
    database $DB$, search parameter $k$, BERT encoder $BERT\_Enc$
    and all-MLP decoder $MLP\_Dec$.
Output: Predicted length $\{\hat{L}_j\}_{j=1,\dots,n}$
1: while true do
2:     for all $j < n$ do
3:         $V_j = BERT\_Enc(P_j)$                ▷ Embed input user prompt
4:         $I_j = \text{argmax}_k S(DB, V_j)$        ▷ Database vector search
5:         if $S(I_j) < s_0$ then   ▷ Case I: No similar vectors found
6:             $\hat{L}_j = MLP\_Dec(I_j)$
7:         else                  ▷ Case II: Similar vector(s) found
8:             $\hat{L}_j = \sum S(I_j) \cdot DB(I_j)$             ▷ Aggregate results
9:         end if
           return $\hat{L}_j$
10:        $DB.update(V_j, L_j)$                          ▷ Update database
11:    end for
12: end while
```

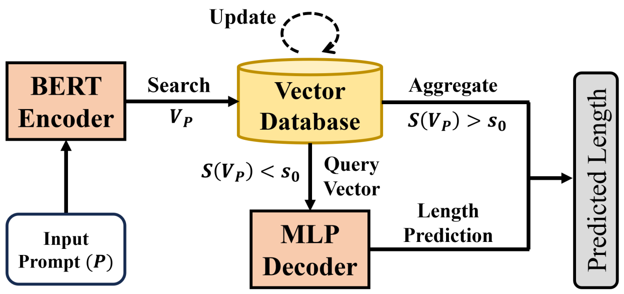

Figure 4: Retrieval-based Length Predictor Architecture.

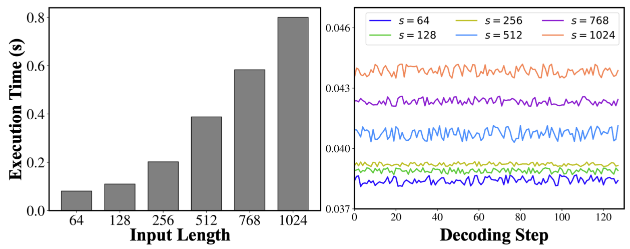

Figure 5: Execution breakdown of the OPT-13B. The left figure shows the prefill execution time with different input lengths (s), and the right figure shows the execution time for different decoding steps.

DB, the query context is fed to an all-MLP decoder fine-tuned for regression tasks to predict the sequence length. After each request is finished, we update the vector DB with new queries and their actual response length to keep the dataset current and relevant. The details of this procedure are formulated as in Algorithm 1.

To evaluate the effectiveness of our retrieval-based method, we construct our database using the OpenChat [32] dataset and fine-tune the decoder on the Alpaca [27] and ShareGPT [28] dataset. Our approach achieves an average prediction error of 3.4% and 9.2% between predicted lengths and actual lengths on two datasets respectively, which is 1.4× and 1.6× lower than proxy-based methods. Moreover, thanks to the adoption of a query database, our method can reduce the prediction latency by up to 70% by avoiding unnecessary computation. This demonstrates that our speculative model can accurately predict output sequence length with low overhead.

Predicting Execution Runtime. Given the output length, we can thus formulate the execution of an inference job using a mathematical model. We first dissect the inference process into two parts, e.g. prefilling and decoding, and use the following to denote total execution time:

$$T_{gen}(s, n) = T_{pre}(s) + T_{dec}(s, n) \tag{3}$$

where s denotes the input length, and n denotes the output length. For the same hardware and model, the prefilling latency depends on the input sequence length, and the decoding latency is determined by both input and output length.

To further study the relationship between the sequence length and the execution latency of each stage, we perform an inference benchmark for OPT-13B under different input sequence settings. As shown in Figure 5, we can see that 1) the prefilling execution time $T_{pre}(s)$ increases almost linearly with input length s. Coincidentally, the decoding latency exhibits a similar linear relationship concerning input length s, and the decoding latency $T_{dec}(s, n)$ for different iterations does not exhibit any noticeable changes, largely due to the KV cache. Based on these observations, we propose to estimate the prefilling and decoding latency as the following:

$$T_{pre}(s) \approx s \cdot T_0 \tag{4}$$
$$T_{dec}(s, n) \approx n \cdot (\alpha s + \beta) \tag{5}$$

where $\{T_0, \alpha, \beta\}$ are the linear regression coefficients that can be determined by profiling numerous samples of prefilling latency of different input and output sequence lengths.

Preemptive Scheduling. Leveraging the above analytical model that can accurately estimate the execution latency of LLM inference, we design a simple scheduler using priority queues [11]. The scheduler manages several job queues and assigns the priority order based on the estimated remaining execution time at the iteration level. We employ a virtual aging [26] mechanism for jobs in the lower priority queues to prevent starvation. To handle short mispredictions, we demote the job to the lower priority queue if it exceeds its predicted length and doubles the predicted length. With our proposed priority queues, our speculative scheduler can dynamically adjust the priority order for each job, and manage heterogeneous workloads more efficiently.

3.2 Adaptive Memory Management

Our proposed speculative scheduler provides iteration-level preemption to manage job scheduling orders, however, directly applying our methodology to LLM inference serving encounters the obstacle of additional memory overhead. In FCFS, we only need to store the intermediate KV cache for the executed jobs in the batch. In contrast, with preemptive scheduling, more jobs are in pending states in the GPU, and additional memory space must be allocated to store their KV cache for future use. Determined by the model size, batch size, and sequence length, the KV cache could consume significant memory space, potentially causing out-of-memory (OOM) errors, and halting the execution [25, 39].

A straightforward solution to mitigate the memory overhead issues is to defer all the newly arrived jobs in the priority queue when the GPU memory reaches the maximum capacity. However, this solution loses its validity when incoming new jobs have higher priority (shorter execution time), but are blocked due to memory space limitation. In resource-constrained settings, such as single GPU-CPU systems, our speculative scheduling would be downgraded to FCFS, and the HoL blocking issues would resurface. Another practical solution is to perform deletion for the unused KV cache for jobs in the queues to make room for the new jobs. But this solution suffers from two major downsides: 1) the deleted KV cache has to be recomputed when low-priority jobs are scheduled for execution, which induces additional compute latency; 2) with the aging mechanism, the killed low-priority jobs would be promoted to the high-priority queue after passing the age threshold, which then kills the currently executed jobs, leading to potential deadlocks.

Algorithm 2 Dynamic Swapping

```
Input: Priority job queue $\{Q_i\}_{i=1,...,h}$, GPU memory $GM$, CPU
    memory $CM$, GPU job limit $M$.
 1: while true do
 2:     for $q \in \{Q_1, ..., Q_h\}$ do
 3:         EWT($q$).$sort()$
 4:         for $i = 1$ to $len(q)$ do
 5:             if $J_i$ not in GPU and $i < M$ then
 6:                 $CM$.upload($J_i$)            ▷ Preemptive upload
 7:             else
 8:                 $GM$.offload($J_i$)           ▷ Preemptive offload
 9:             end if
10:             if $len(q) < M$ then             ▷ Update GPU job limit
11:                 $M = M - len(q)$
12:             end if
13:         end for
14:     end for
15: end while
```

Dynamic Swapping. Previous works have explored leveraging the memory of different hierarchies to accelerate LLM inference in resource-constrained environments, however, they are mostly for offline batched inference and schedule the KV cache swapping at either head or token level [25, 39]. In this work, we design a priority-based KV cache manager that dynamically swaps the KV cache for preempted jobs between GPU HBM and CPU memory at the request level. Specifically, ALISE performs dynamic swapping operations of the KV cache based on the metric of estimated wait time (EWT). Generally, jobs with higher EWT tend to be offloaded to the CPU DRAM to make space for high-priority jobs, while jobs with lower EWT tend to be uploaded for later execution. Additionally, the offloading and uploading operations are overlapped with computation to avoid additional latency.

For a job with priority p, we can calculate the EWT by summing all the execution time of jobs with higher priority as follows:

$$EWT(J) = \sum_{m<p}^{m} Predictor(J) \tag{6}$$

Due to the existence of the aging mechanism (age threshold K), we also need to compare the time needed to promote J_i to the high-priority queue for execution. Therefore, we reformulate Equation 6 as follows:

$$EWT(J) = \min(\sum_{m<p}^{m} Predictor(J), T_{promote}(J, K)) \tag{7}$$

Leveraging the above EWT definitions, we can decide the order of offloading and uploading of each job by setting the maximum number of jobs allowed in the GPU, which is constrained by the total available GPU memory space. The details of our dynamic swapping is shown in Algorithm 2. For each priority queue, we first calculate and sort the EWT of all the jobs in the queue (lines 2-3). We keep the preempted jobs in the GPU if not exceeding the GPU job limit M, and perform the preemptive upload of the KV cache if not in the GPU (lines 5-6). For jobs exceeding the job limit, we perform preemptive offload operations (lines 7-8). The GPU limit M is also updated by each queue accounting for the size of previous queues (lines 10-11).

KV Compression. Previous works have utilized quantization to accelerate attention computation by compressing model weights [9, 14]. In this work, we leverage quantization for a different purpose, i.e., compressing the unused KV cache to reduce memory overhead. We adopt a fine-grained channel-wise quantization for the KV cache to ensure robust performance and minimal information loss. More specifically, we use the following formula to quantize KV cache to b-bit integers in memory and de-quantize them to their original format (FP16 in this work) for computation:

$$x_{\mathrm{q}} = \mathrm{round}(\frac{1}{\lambda}x + z), \quad x = \lambda(x_{\mathrm{q}} - z) \tag{8}$$

where zero point $z = \lfloor(\frac{-2^b}{max-min})\rceil$, and the scaling factor $\lambda = \frac{max-min}{2^b-1}$. In this work, we choose the 8-bit integer for quantizing the KV cache to ensure minimal degradation on text generation quality.

3.3 Implementation

ALISE is implemented on top of PyTorch [20] and HuggingFace Transformer libraries [33] with 4.5K lines of Python. Our execution engine is based on vLLM [13], which we modify to support iteration-level preemptive scheduling and interact with the KV cache manager for dynamic swapping. ALISE also features customized fused kernels for LayNorm, Attention, and ReLU operations, just like other systems for transformer-based LLMs [13, 16, 35]. We implement these kernels based on python-based Triton compiler [29]. To increase the compute utilization of GPUs and improve inference efficiency, we adopt the iteration-level continuous batching technique [35], where we apply batching only to a handful of operations, such as Linear, LayerNorm, and ReLU, which are compatible with irregularly shaped tensors.

4 EVALUATION

4.1 Experimental Setup

Hardware. We conduct our experiments on a single customized high-performance GPU instance, configured with two NVIDIA Tesla V100 GPUs with 32 GB VRAM, and a 2.60 GHz Intel Xeon CPU connected over PCIe 4.0×16 and 1024 GB DRAM.

Workloads. To emulate the LLM workloads, we synthesize traces of user requests based on the Alpaca [27] and ShareGPT [28] dataset since there is no publicly available request trace for LLMs. The Alpaca dataset is a GPT-3.5 generated instruction dataset and is widely used for LLM inference evaluation [13, 40]. The ShareGPT dataset is a public dataset that collects the chatbot conversation from ChatGPT users, which exhibit greater variance in contents and sequence length, as shown in Figure 7. We tokenize each dataset and use their input and output lengths to simulate real-world user requests. In terms of request timestamps, we generate the arrival time based on the Poisson distribution.

Models. Following prior works on LLM serving, we use the popular OPT [37], one of the most representative open-sourced LLM families widely used in both industry and academia, for our main evaluation. We choose three model size configurations, namely 2.7B,

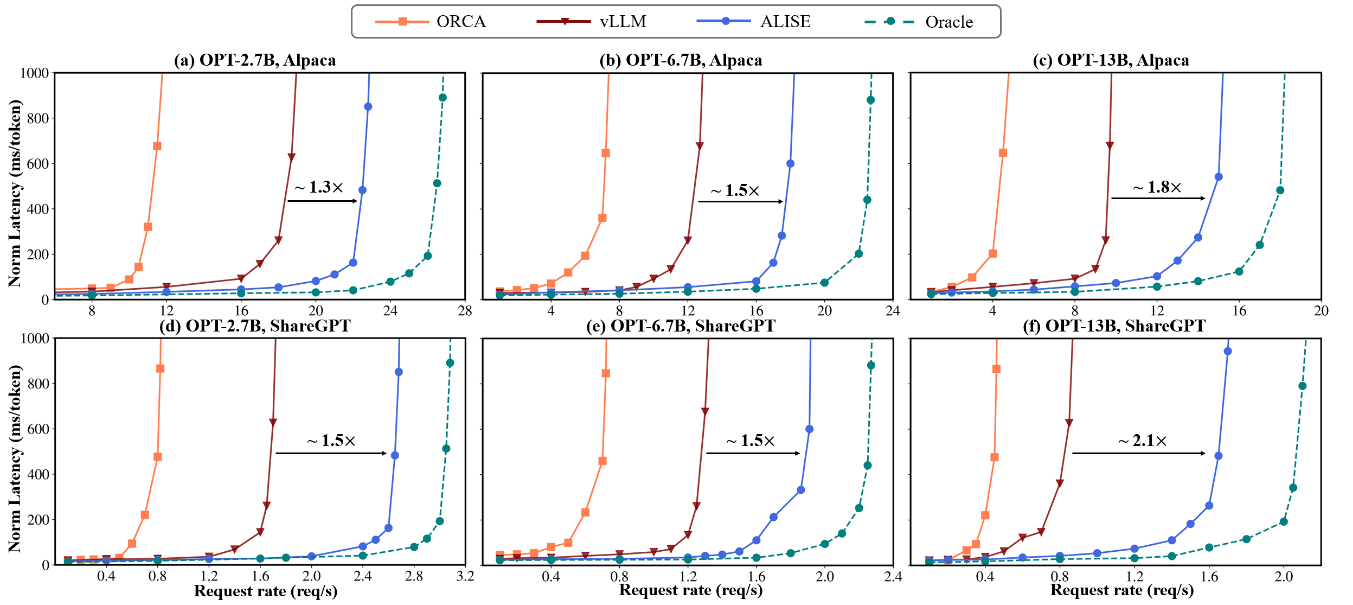

Figure 6: End-to-end performance comparison of ALISE and baselines, including ORCA [35], vLLM [13], and the Oracle on the Alpaca [27] and ShareGPT dataset [28].

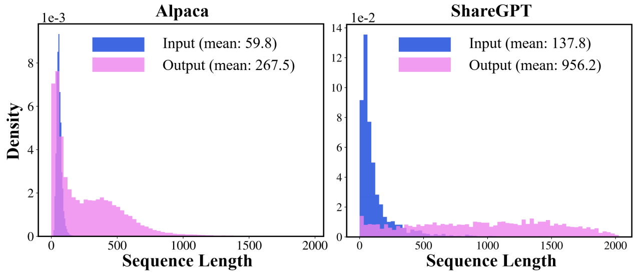

Figure 7: Input and output length distribution of the Alpaca and ShareGPT datasets.

6.7B, and 13B. Table 1 lists the detailed model and corresponding GPU configurations. For all experiments, we maintain the models in the GPU HBM during serving and use FP16 precision for model weights, activations, and INT8 for the KV cache.

Baselines. We compare ALISE with three baselines.

- **ORCA** [35]: ORCA is one of the first LLM systems to adopt the iteration-level first-come-first-serve (FCFS) scheduling at the iteration level.
- **vLLM** [13]: vLLM is the state-of-the-art online LLM inference serving system, which manages the KV cache at the block level but still uses FCFS protocol.
- **Oracle**: We also implement an ideal system baseline with a perfect predictor that knows the output lengths, which we term Oracle, to show the upper bound of our system performance.

Metrics. To measure the inference serving throughput, we use the normalized latency of the systems, defined as the mean of the request's end-to-end latency divided by the number of generated tokens. A high-throughput serving system should retain low normalized latency against high request rates. We set the batch size as large as possible according to the GPU memory capacity. All system performances are evaluated with 30-minute traces.

Table 1: Model configurations.

Model	# Heads	# Layers	Hidden Size	Parameter Size
OPT-2.7B	32	32	2560	5 GB
OPT-6.7B	40	40	5120	13 GB
OPT-13B	40	40	5120	24 GB

4.2 End-to-End Performance

The first row of Figure 6 shows the results on the Alpaca dataset. We can see that as the request rate increases, the normalized latency gradually increases but then explodes after a certain threshold. Such observation aligns with previous works [13, 25, 35, 39], where LLM inference is often *memory-bound*, and once the request rate reaches the memory capacity of the serving system, the response latency grows infinitely as the request queue pool continues to expand. On the Alpaca dataset, ALISE obtains 1.3 ∼ 1.8× higher throughput compared to vLLM while maintaining similar latency results. This is due to ALISE's speculative scheduling strategy that dynamically adjusts the priority order of each request thus reducing the response delay. For instance, as shown in Figure 6(b), under the latency constraints of 200 ms, ALISE processes 1.5× more requests than vLLM. When compared against ORCA, ALISE sustains up to 3.1× higher request rates. The advantage of ALISE is more pronounced on the ShareGPT dataset, as shown in the second row of Figure 6. This is because the ShareGPT dataset contains longer input prompts and

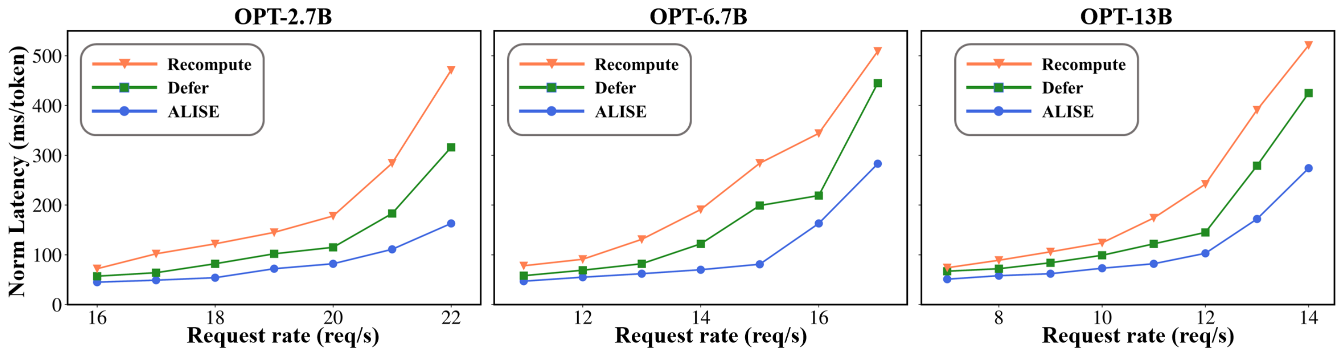

Figure 8: Latency comparison of different memory management protocols in ALISE on the Alpaca dataset.

Table 2: Accuracy and throughput evaluation of our proposed retrieval-based length predictor on the ShareGPT dataset.

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×

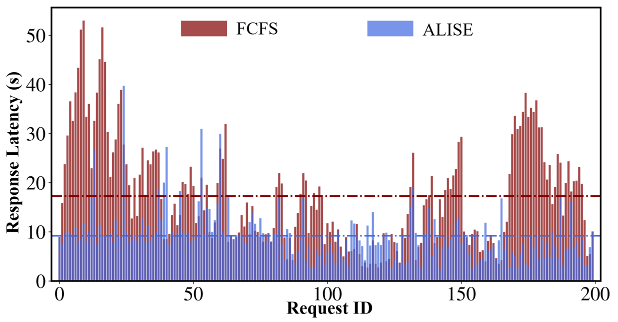

Figure 9: Response latency for 200 sampled requests when serving OPT-13B on the ShareGPT dataset at 2 req/s. The dotted line indicates the mean response latency.

outputs on average than the Alpaca dataset, with much higher variance. ALISE achieves up to 2.1× throughput improvement against the state-of-the-art vLLM. Compared to ORCA, ALISE also obtains up to 4.3× high throughput under similar latency constraints.

4.3 Performance Analysis

Impact of KV Cache Management. We further demonstrate the benefits of our proposed adaptive KV cache management by comparing our design with two strawman solutions, namely *Recompute* and *Defer* strategy. The *Recompute* strategy deletes the KV cache of preempted low-priority jobs and recomputes the KV cache when jobs are rescheduled for execution, while the *Defer* strategy simply defers newly arrived jobs when the GPU does not have enough memory space for executing the job.

We perform the experiments on the OPT models with varying request rates on the Alpaca dataset. As shown in Figure 8, when the request rate is relatively low, the gap between ALISE and the other two methods is not significant, as the KV cache for most jobs is in the GPU. However, as the request rate increases, the gap between these solutions is widened: ALISE outperforms *Defer* and *Recompute* by up to 1.8× and 2.8× in terms of normalized latency under the same request rate. For the *Defer* strategy, with a higher request rate, the HoL blocking issues resurface, as more newly arrived jobs with shorter execution times are delayed, thus leading to increased response latency; for the *Recompute* strategy, the increased latency is largely attributed to the increased execution time for token generation, since for scheduled preempted jobs, the system has to recompute the KV cache of previous iterations to generate the subsequent tokens.

Impact of Length Predictor. To further show the effectiveness of our proposed retrieval-based length predictor, we conduct experiments to compare the accuracy and throughput performance with previous proxy-based predictors [12, 22] on the ShareGPT dataset. To measure the prediction accuracy, we categorize the lengths into evenly divided bins, and we regard the prediction as accurate when both lengths fall into the same bucket. In Table 2, we can see that our method improves previous methods in both aspects. Specifically, thanks to the introduction of a query database storing history user information, a retrieval-based approach can predict new queries with high accuracy and low prediction error; as some queries do not need to go through the predictor model, our method also achieves significant speedup against proxy-based methods regarding prediction latency. Moreover, with more accurate length information, the scheduler can assign more appropriate priority orders accordingly, improving the overall inference throughput performance.

Response Latency Analysis. To better understand our speculative scheduler on the system throughput, we randomly sample 200

Table 3: Throughput improvement for different models on the ShareGPT and Alpaca datasets, measured in processed requests per second.

	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%)

consecutive generated requests during serving OPT-13B on the ShareGPT dataset and compare the end-to-end response latency of each request when served with non-preemptive FCFS scheduling and ALISE. As shown in Figure 9, our system significantly reduces the response latency for most requests, thanks to the preemption of speculative scheduling, which alleviates the head-of-line blocking issues. ALISE also achieves close to 46% reduction in mean response latency against FCFS baselines.

Results on More Models. To verify the effectiveness of our method on different models, we choose several other popular open-sourced LLMs, such as LLaMA-7B/13B [30] and Pythia-12B [2] for further evaluation. As shown in Table 3, ALISE provides consistent improvement over previous baseline systems for these LLMs.

5 RELATED WORK

DNN Inference Serving. Recent years have witnessed the exponential of AI-driven applications in real-world practices, and numerous dedicated systems [5, 10, 23, 36] have been developed to meet such growing demand. They generally focus on batching [5], scheduling [10], and caching optimizations [36] for serving small-scale models such as ResNet. More recently, INFaaS [23] automates the model selection and deployment process by navigating the best trade-off between model variants and performances. However, these systems fail to take into account the multi-iteration characteristics and the unpredictability of LLM inference, resulting in missed opportunities for further optimizations.

LLM Inference Serving. Several specialized inference serving systems have been developed for Transformer-based LLMs [13, 16, 35]. FasterTransformer is among the first to optimize training and inference serving for large transformer-based LLMs by utilizing GPU kernel optimizations and model parallelism [16]. ORCA further develops iteration-level scheduling and selective batching by leveraging the multi-iteration nature of LLM inference [35]. vLLM proposes PagedAttention to store the KV cache in a non-contiguous paged memory to alleviate memory fragmentation [13]. While they have achieved promising results, existing LLM systems use the non-preemptive first-come-first-served (FCFS) [38] to process LLM inference workloads. It is known that such run-to-completion scheduling exhibits head-of-line blocking since once a workload is scheduled, it must run until it finishes. An LLM inference job with a long output length would run for a long time to block following relatively short jobs, thus causing head-of-line blocking and increasing the response latency. Given the heterogeneous nature of real-world requests, the inflexible FCFS scheduling has become the new bottleneck for low-latency and high-throughput LLM serving systems.

LLM Output Length Prediction. There are several limited works on predicting the LLM output sequence length. S^3 adopts a DistillBERT [24] model to classify the output length range given LLM input queries, while SSJF [22] employs a similar proxy model to perform classification and regression for sequence length. Instruction tuning has also been leveraged to prompt LLM itself to output sequence length, but it generally introduces additional bias and requires intensive engineering efforts [19, 40]. In our work, we develop a retrieval-based predictor building upon a vector database, which provides more accurate length prediction at low latency overhead.

Memory Optimization for LLMs. Numerous techniques have been proposed to reduce the memory overhead of LLM inference. SparseTransformer accelerates the inference process by generating sparsity patterns with a fixed stride on all sequence tokens [4]. FlashAttention aims to reduce memory accesses between on-chip SRAM and off-chip HBM in GPUs with fine-grained tiling and partitioning at the kernel level [6, 7]. FlexGen focuses on resource-constrained single GPU-CPU scenarios and formulates a static offloading strategy for KV cache, model weights, and intermediate results utilizing CPU DRAM and secondary storage [25]. ALISA proposes an algorithm-system co-design approach that exploits the sparsity patterns in LLM inference to achieve the optimal trade-offs between caching and recomputation [39]. Orthogonal to existing memory optimization techniques, our work focuses on scheduling optimization to minimize response latency for serving online heterogeneous workloads.

6 CONCLUSION

In this work, we present a new efficient LLM inference serving solution, termed ALISE, to address the head-of-line (HoL) blocking issues in the existing FCFS scheduling strategy. ALISE leverages a novel retrieval-based speculative to schedule incoming jobs by their remaining execution time preemptively, thus minimizing potential queuing delays for heterogeneous workloads. To alleviate the memory overhead of preempted jobs, we design adaptive memory management that dynamically performs swapping operations for unused KV cache and quantization-based compression. Experiments demonstrate that ALISE obtains up to 1.8× and 2.1× throughput improvement over the state-of-the-art systems, such as vLLM, on the Alpaca and ShareGPT datasets, respectively.

ACKNOWLEDGEMENT

This work was sponsored in part by the U.S. National Science Foundation (NSF) under Grants 1907765 and 2400014. The authors would like to thank the anonymous ICCAD reviewers for their constructive feedback to improve this work.

REFERENCES

[1] Meta AI. 2024. Introducing Llama 3.1: Our most capable models to date. https://ai.meta.com/blog/meta-llama-3-1/

[2] Stella Rose Biderman, Hailey Schoelkopf, Quentin G. Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *ArXiv* abs/2005.14165 (2020).

[4] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. *ArXiv* abs/1904.10509 (2019).

[5] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.

[6] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *ArXiv* abs/2307.08691 (2023).

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv* abs/1810.04805 (2019).

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *International Conference on Learning Representations (ICLR)* (2023).

[10] Arpan Gujarati, Reza Karimi, Safya Alzayat, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *USENIX Symposium on Operating Systems Design and Implementation.*

[11] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. 2019. Priority queues. *The Art of Multiprocessor Programming* (2019).

[12] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S3: Increasing GPU Utilization during Generative Inference for Higher Throughput. *Advances in Neural Information Processing Systems* 36 (2023).

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Haotong Zhang, and I. Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles* (2023), 611–626.

[14] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *ArXiv* abs/2306.00978 (2023).

[15] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations.*

[16] NVIDIA. 2019. FasterTransformer. https://github.com/NVIDIA/FasterTransformer

[17] OpenAI. 2022. Introducting ChatGPT. https://openai.com/blog/chatgpt

[18] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *North American Chapter of the Association for Computational Linguistics.* 6151–6162.

[19] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv* abs/2203.02155 (2022).

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[21] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023).

[22] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2024. Efficient Interactive LLM Serving with Proxy Model-based Sequence Length Prediction. *ArXiv* abs/2404.08509 (2024).

[23] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[24] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv* abs/1910.01108 (2019).

[25] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark W. Barrett, Joseph Gonzalez, Percy Liang, Christopher Ré, Ioan Cristian Stoica, and Ce Zhang. 2023. High-throughput Generative Inference of Large Language Models with a Single GPU. In *International Conference on Machine Learning*. PMLR, 31094—-31116.

[26] Abraham Silberschatz, Greg Gagne, and Peter Galvin. 2004. Operating System Principles.

[27] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[28] ShareGPT Team. 2023. ShareGPT. https://sharegpt.com/

[29] Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (2019).

[30] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *ArXiv* abs/2307.09288 (2023).

[31] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *ArXiv* abs/1706.03762 (2017).

[32] Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. 2023. OpenChat: Advancing Open-source Language Models with Mixed-Quality Data. *ArXiv* abs/2309.11235 (2023).

[33] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).

[34] Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920* (2023).

[35] Gyeong-In Yu and Joo Seong Jeong. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *USENIX Symposium on Operating Systems Design and Implementation.*

[36] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the Wild. In *Symposium on Networked Systems Design and Implementation.*

[37] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *ArXiv* abs/2205.01068 (2022).

[38] Wei Zhao and John A. Stankovic. 1989. Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems. *[1989] Proceedings. Real-Time Systems Symposium* (1989), 156–165.

[39] Youpeng Zhao, Di Wu, and Jun Wang. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. *ArXiv* abs/2403.17312 (2024).

[40] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline. *ArXiv* abs/2305.13144 (2023).