DYNAMIC LORA SERVING SYSTEM FOR OFFLINE CONTEXT LEARNING

Shiyi Cao [1] **Sijun Tan**

ABSTRACT

The "pretrain-then-finetune" paradigm is commonly adopted in the deployment of large language models. Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method, is often employed to adapt a base model to a multitude of tasks, resulting in a substantial collection of LoRA adapters derived from one base model. In this project, we try to address two problems: (1) how to derive LoRA adapters from long-context documents so that we can have tailored finetuned adapters per user, and (2) how to design a system for scalable serving of many LoRA adapters concurrently. To address (1), we propose offline context learning, a novel strategy to memorize long documents through parameter-efficient finetuning. Our evaluation on the QuALITY QA dataset shows that our proposed method outperforms in-context learning while processing over 32x fewer tokens during inference. To resolve (2), we propose S-LoRA, a serving system scalable to thousands of LoRA adaptors on a single GPU. Compared to state-of-the-art libraries such as HuggingFace PEFT and vLLM (with naive support of LoRA serving), our serving system S-LoRA can improve the throughput by up to 4 times and increase the number of served adapters by several orders of magnitude.

1 INTRODUCTION

Large language models (LLMs) have exhibited exceptional abilities to perform a variety of language-processing tasks. Recent studies have revealed that the way the prompt is crafted can markedly influence the performance of LLMs on downstream tasks. Stemming from this insight, the community has embraced the paradigm of in-context learning (Dong et al., 2023), which requires carefully designing prompts to improve the performance of LLM on downstream tasks. As a consequence of this new paradigm, prompts become increasingly longer. Many common downstream tasks, such as document summarization, document question answering, code completion, and chatbots requires LLMs to process over thousands of tokens.

However, scaling LLMs to long context still presents several key challenges. First, due to the self-attention mechanism, transformer-based LLMs incur a quadratic computational and memory overhead as sequence length increases. Many long-context tasks require repeatedly processing over the same prompt, which incurs extra latency overhead and substantial costs, as most commercial LLMs operate on a pricing model that is directly tied to the number of tokens processed. Second, even though recent LLMs (e.g. GPT4-turbo, Claude) could support input sequence length to as long as 100k tokens, studies have shown that the performance of LLMs degrades as the input length increases. For example, (Liu et al., 2023a) reveals that LLMs exhibit significantly better performance when the relevant information is positioned either at the beginning or end of the context window.

To address the long context challenge, we propose long-context learning – a novel approach based on LoRA finetuning to learn these contexts offline. To illustrate our idea, consider this analogy: envision an LLM as a student preparing for an exam, where we are the examiners providing study materials and questions. Traditional in-context learning resembles an open-book exam, where the LLM can refer to materials while answering questions. In contrast, we propose a method akin to a closed-book exam, where the LLM studies the material beforehand and then answers questions without accessing the material during the exam. In machine learning terms, this involves using parameter-efficient finetuning methods to enable the LLM to learn long contexts offline. Then, during inference, we use the finetuned model weights without needing to prepend the relevant long context to the prompt. Since each sets of documents is fine-tuned with a different LoRA adaptor, the main challenge that arises from this setting is how we could efficiently swap and serve multiple LoRA adaptors at the same time.

When scaling the fine-tuning of a base model for tasks such as offline context learning, the associated training and serving costs can become substantial. To address this, several parameter-efficient fine-tuning methods have been developed. A prime exemplar is Low-Rank Adaptation

[1]The serving system is developed by the S-LoRA team: https://arxiv.org/abs/2311.03285 .

(LoRA) (Hu et al., 2021), which enables efficient fine-tuning by updating only low-rank additive matrices. These matrices consist of a small number of parameters, referred to as adapter weights. LoRA has shown that by fine-tuning just these adapter weights, it is possible to achieve performance on par with full-weight fine-tuning. However, despite considerable research into fine-tuning, the question of how to serve these fine-tuned variants at scale remains unexplored.

The original LoRA paper proposes swapping adapters by adding and subtracting LoRA weights from the base model for multi-model serving. While this approach enables low-latency inference for a single adapter and serial execution across adapters, it significantly reduces overall serving throughput and increases total latency when serving multiple adapters concurrently.

We observe that the shared base model, which underpins numerous LoRA adapters, presents a substantial opportunity for batched inference. To achieve high-throughput multi-adapter serving, it is advantageous to separate the batchable base model computation from individual LoRA computations.

In summary, to provide highly usable and efficient personalized LLM service, we will need to address the following challenges: First, serving many LoRA adapters simultaneously requires efficient memory management and smarter batched operations. Since GPU memory is limited, we must store adapter weights outside the GPU and dynamically fetch them when needed. However, dynamically loading and unloading adapters of varying sizes, coupled with the dynamic allocation and deallocation of KV cache tensors for requests with different sequence lengths, can lead to significant memory fragmentation and I/O overhead. Moreover, the batched computation of many adapters with distinct ranks in non-contiguous memory is challenging and demands the development of new computation kernels.

The primary contributions of this project are summarized as follows:

- a novel strategy called offline context learning to memorize long documents prior to inference using LoRA finetuning.
- a scalable system for serving a large number of LoRA adapters concurrently with techniques such as unified paging and heterogeneous batching.

We evaluate our approach on the QuALITY long-context QA dataset, and our preliminary results show that our offline learning method outperforms in-context learning while processing over 32x fewer tokens during inference.

We also evaluate the serving system by serving Llama-7B/13B. Results show that we can serve thousands of LoRA adapters on a single GPU with a small overhead. When compared to the state-of-the-art parameter-efficient fine-tuning library, Huggingface PEFT, we can enhance throughput by up to $30\times$. In comparison to the high-throughput serving system vLLM using a naive support of LoRA serving, we can improve throughput by up to $4\times$ and increase the number of served adapters by several orders of magnitude.

2 Background

2.1 LoRA Finetuning

Low-Rank Adaptation (LoRA) (Hu et al., 2021) is a parameter-efficient fine-tuning method designed to adapt pre-trained large language models to new tasks. The motivation behind LoRA stems from the low intrinsic dimensionality of model updates during adaptation. In the training phase, LoRA freezes the weights of a pre-trained base model and adds trainable low-rank matrices to each layer. This approach significantly reduces the number of trainable parameters and memory consumption. When compared to full parameter fine-tuning, LoRA can often reduce the number of trainable parameters by orders of magnitude (e.g., $10000\times$) while retaining comparable accuracy. For the inference phase, the original paper suggests merging the low-rank matrices with the weights of the base model. As a result, there is no added overhead during inference, setting it apart from previous adapters like (Houlsby et al., 2019) or prompt tuning methods such as (Lester et al., 2021).

Formally, for a pre-trained weight matrix $W \in \mathbb{R}^{h\times d}$, LoRA introduces the update as $W' = W + AB$, where $A \in \mathbb{R}^{h\times r}$, $B \in \mathbb{R}^{r\times d}$, and the rank $r \ll \min(h, d)$. If the forward pass of a base model is defined by $h = xW$, then after applying LoRA, the forward pass becomes

$$h = xW' = x(W + AB) \tag{1}$$
$$= xW + xAB. \tag{2}$$

Typically, this adjustment is only applied to the query, key, value, and output projection matrices in the self-attention module, excluding the feed-forward module.

Because LoRA greatly reduces the training and weight storage costs, it has been widely adopted by the community, and people have created hundreds of thousands of LoRA adapters for pre-trained large language models and diffusion models (Mangrulkar et al., 2022).

2.2 Serving Large Language Models

Most large language models (LLMs) are based on the transformer architecture (Vaswani et al., 2017). The number of parameters in an LLM ranges from several billion to several trillion (Brown et al., 2020; Chowdhery et al., 2022; Fedus et al., 2022), corresponding to disk sizes spanning several gigabytes to even terabytes. This scale results in LLM serving having significant computational and memory demands.

Additionally, the inference process for LLMs requires iterative autoregressive decoding. Initially, the model carries out a forward pass to encode the prompt. Following this, it decodes the output one token at a time. The sequential process makes decoding slow. Since each token attends to the hidden states of all its preceding tokens, it becomes essential to store the hidden states of all previous tokens. This storage is referred to as the "KV cache". Such a mechanism adds to the memory overhead and causes the decoding process to be more memory-intensive than computation-intensive.

The challenges become even more pronounced in online settings, where requests of varying sequence lengths arrive dynamically. To accommodate such dynamic incoming requests, Orca (Yu et al., 2022) introduces a method of fine-grained, iteration-level scheduling. Instead of scheduling at the request level, Orca batches at the token level. This approach allows for the continuous addition of new requests to the currently running batch, resulting in substantially higher throughput. vLLM (Kwon et al., 2023) further optimizes Orca's memory efficiency using PagedAttention. PagedAttention adopts concepts from virtual memory and paging in operating systems and manages the storage and access of dynamic KV cache tensors in a paged fashion. This method efficiently reduces fragmentation, facilitating larger batch sizes and higher throughput.

When serving very large models that exceed the memory capacity of a single GPU, or when there are stringent latency requirements, it is necessary to parallelize the model across multiple GPUs. Several model parallelism methods have been proposed, such as tensor parallelism (Shoeybi et al., 2019), sequence parallelism (Korthikanti et al., 2023), pipeline parallelism (Huang et al., 2019), and their combinations (Narayanan et al., 2021; Zheng et al., 2022).

3 Offline Context Learning

In this section, we explain our method on learning compressed representation of long contexts. Our goal is to compress a context of long sequence length (e.g. 4096 tokens) into a much more condensed set of *memory tokens* of finite length (e.g. 128 tokens). Intuitively, the memory tokens should serve as a concise summary and capture the essence of the longer context. We can then use these memory tokens instead of the original long context for downstream instruction-following tasks such as question answering via supervised fine-tuning (SFT). We now provide details on our proposed model and training strategies.

3.1 Model Architecture

We illustrate our proposed model architecture in Figure 1. Concretely, our model has two main components: a context encoder that compresses the original long context (e.g. a long article) into memory tokens, and an LLM decoder that autoregressively outputs given the memory tokens and potentially other prompts such as questions about the context. The context encoder is an LLM encoder that converts the context $\mathbf{c} = \{x_1, \ldots, x_L\}$ where L is the context length into memory tokens $\mathbf{m} = \{m_1, \ldots, m_k\}$, where k is the number of memory tokens and $L \gg k$. In our experiments, we explore two choices for the context encoder: (1) a pretrained context encoder from ICAE (Ge et al., 2023) that contains trainable LoRA weights, (2) a pretrained sentence embedding model such as GTR (Ni et al., 2021) followed by a trainable linear layer that projects the memory embedding dimension to the hidden dimension of the target LLM. The former choice adapts an existing work that is specifically trained to compress long contexts but requires training of the LoRA modules in the model. The latter choice uses a frozen off-the-shelf embedding model and only requires training of a simple linear layer. This is a lot more flexible as we can easily swap between embedding models and the linear layer is considerably simpler to train than LoRA weights. We conduct experiments on both settings and present findings in the next section. For the decoder, we adopt a LoRA-adapted LLM decoder such as LLaMA2-7B (Touvron et al., 2023a). Conditioned on the memory tokens, the decoder is expected to both maintain knowledge of the long context and generate reasonable responses (answers) regarding this context if given additional prompts (questions). We now present the training procedures for our proposed model.

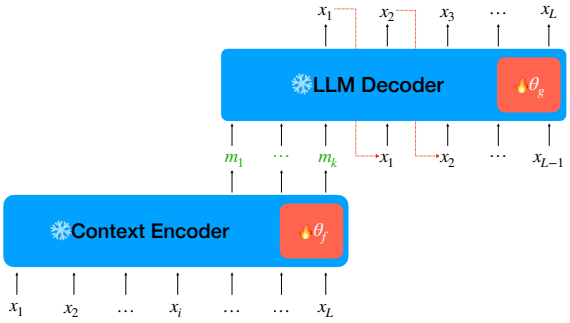

Figure 1. Our proposed model architecture. The context encoder and LLM decoder are both adapted from pretrained models with trainable parameters denoted as θ_f and θ_g respectively. Depending on the design choices of the context encoder (detailed in Section 3.1), θ_f can be modeled with a LoRA adapter or a linear layer. The context encoder maps the original (long) context $\mathbf{c}$ into a compact set of memory tokens $\mathbf{m}$. Given the memory tokens, the decoder can autoregressively reconstrcuct the original context.

3.2 Pretraining via Reconstruction

At the first training stage, our model is trained to compress long contexts into memory tokens that can faithfully repre-

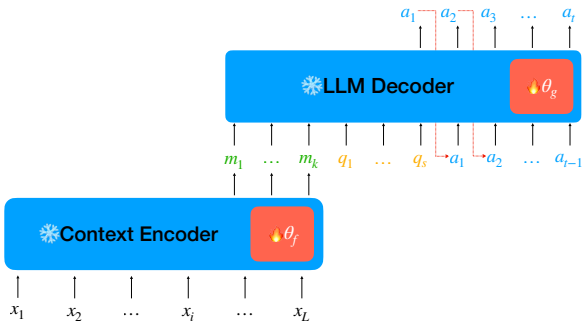

Figure 2. Supervised fine-tuning on instruction data. Given the memory tokens $\mathbf{m}$ and prompts $\mathbf{q}$, the decoder now autoregressively outputs a response $\mathbf{a}$.

sent the meaning of the original contexts. To this end, we condition the LLM decoder on the memory tokens $\mathbf{m}$ to produce the original sequence $\mathbf{c}$ and train the model via a reconstruction loss (illustrated in Figure 1):

$$\mathcal{L}_{rec} = \max_{\theta_f, \theta_g} P(\mathbf{c}|\mathbf{m}; \theta_f, \theta g, \theta_{context}, \theta_{LLM}), \quad (3)$$

where $\theta_{context}$ and θ_{LLM} represent the frozen weights of the context encoder and LLM decoder while θ_f and θ_g represent the trainable weights[1] in the context encoder and LLM decoder, respectively. Intuitively, this objective is analogous to reciting and memorizing the exam material in the test-taker example in Section 1.

3.3 Fine-tuning via Self-Instruct

Once the model becomes capable of memorizing the long contexts, we further fine-tune the model on instructional data. This enables the model to respond to various prompts or questions regarding the context which is now represented by the compressed memory tokens. To achieve this goal, we first generate an instruction dataset in the form of (context, question, answer) triples. These samples can be generated by either the LLM decoder itself or some other LLM models such as GPT-4.

Concretely, conditioned on the memory tokens $\mathbf{m}$ and given the pair of question $\mathbf{q}$ and answer $\mathbf{a}$, the model is optimized via the following supervised fine-tuning objective (illustrated in Figure 2):

$$\mathcal{L}_{sft} = \max_{\theta_f, \theta_g} P(\mathbf{a}|\mathbf{m}, \mathbf{q}; \theta_f, \theta g, \theta_{context}, \theta_{LLM}). \quad (4)$$

[1]For the decoder, the trainable weights are LoRA adapters. For the encoder, the trainable weights can be either LoRA modules or weights of a linear layer, depending on the design choices of the encoder as detailed in Section 3.1.

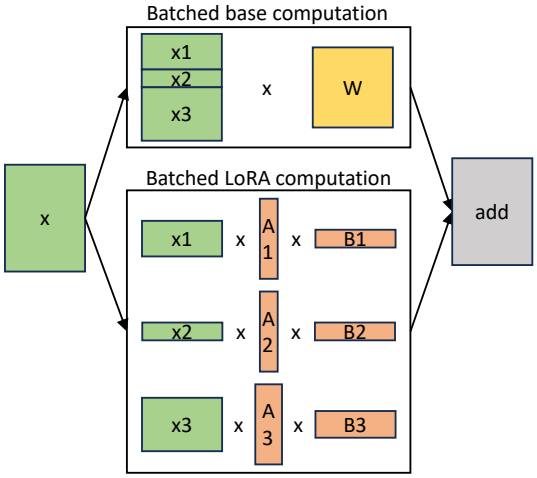

Figure 3. Separated batched computation for the base model and LoRA computation. The batched computation of the base model is implemented by GEMM. The batched computation for LoRA adapters is implemented by custom CUDA kernels which support batching various sequence lengths and adapter ranks.

Intuitively, this objective is analogous to taking mock exam questions in the test-taker example in Section 1.

4 Serving System

Our underlying serving system S-LoRA encompasses three principal components of innovation. In Subsection 4.1, we introduce our batching strategy, which decomposes the computation between the base model and the LoRA adapters. The ability to batch across concurrent adapters, introduces new challenges around memory management. In Subsection 4.2, we generalize PagedAttention (Kwon et al., 2023) to Unfied Paging which support dynamically loading LoRA adapters. This approach uses a unified memory pool to store the KV caches and adapter weights in a paged fashion, which can reduce fragmentation and balance the dynamic changing size of the KV caches and adapter weights.

4.1 Batching

Our batching strategy aims to support online and high-throughput serving of many LoRA adapters simultaneously.

For a single adapter, the method recommended by (Hu et al., 2021) is to merge the adapter weights into the base model weights, resulting in a new model (see Eq. 1). This has the advantage that there is no additional adapter overhead during inference, since the new model has the same number of parameters as the base model. In fact, this was a prominent feature of the original LoRA work.

However, when there are multiple adapters, merging the weights into the base model leads to multiple weight copies and missed batching opportunities. Directly merging the

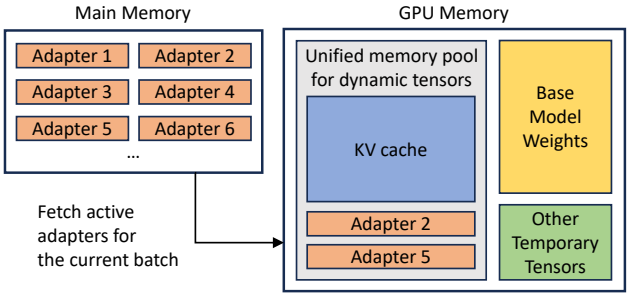

Figure 4. Overview of memory allocation in S-LoRA. S-LoRA stores all adapters in the main memory and fetches the active adapters for the current batch to the GPU memory. The GPU memory is used to store the KV cache, adapter weights, base model weights, and other temporary tensors.

models requires maintaining many copies of the full language model. In the original LoRA paper, the authors proposed adding and subtracting LoRA weights on the fly to enable serving multiple models without increasing the memory overhead. However, this approach doesn't support concurrent inference on separate LoRA adapters and therefore limits batching opportunities.

As illustrated in Subsection 6.3, merging LoRA adapters into the base model is inefficient for the multi-LoRA high-throughput serving setting. Instead, we propose computing the LoRA computation xAB on-the-fly as shown in Eq. 2. This avoids weight duplication and enables batching of the more costly xW operation. But this approach also increases the computation overhead. However, because the cost of xAB is substantially lower than xW and there is a considerable savings from batching xW across different adapters, we show that the savings far exceed the additional overhead.

Unfortunately, directly implementing the factored computation of the base model and individual LoRA adapters using the batch GEMM kernel from the existing BLAS libraries would require significant padding and result in poor hardware utilization. This is because of the *heterogeneity* of sequence lengths and adapter ranks.

In S-LoRA, we batch the computation of the base model and then employ custom CUDA kernels to execute the additional xAB for all adapters separately. This process is illustrated by Figure 3. Instead of naively using padding and using the batch GEMM kernel from the BLAS library for the LoRA computation, we implement custom CUDA kernels for more efficient computation without padding.

While the number of LoRA adapters can be large if we store them in main memory, the number of LoRA adapters needed for the currently running batch is manageable, because the batch size is bounded by the GPU memory. To take advantage of this, we store all LoRA adapters in the main memory and fetch only the LoRA adapters needed for the currently running batch to the GPU RAM when running the inference for that batch. In this case, the maximum number of adapters that can be served is bounded by the main memory size. This process is illustrated by Figure 4. To achieve high-throughput serving, we adopt the iteration-level scheduling batching strategy from Orca (Yu et al., 2022). In this approach, requests are scheduled at the token level. We immediately incorporate a new request into the running batch if space is available. The request will exit the batch once it reaches the maximum number of generated tokens or fulfills other stopping criteria. This process reduces GPU memory usage but introduces new memory management challenges. In Subsection 4.2, we will discuss our techniques to manage memory efficiently.

4.2 Unified Paging

Compared to serving a single base model, serving multiple LoRA adapters simultaneously presents new memory management challenges. To support many adapters, S-LoRA stores them in the main memory and dynamically loads the adapter weights needed for the currently running batch into GPU RAM.

During this process, there are two noticeable challenges. The first is memory fragmentation, resulting from the dynamic loading and offloading adapter weights of various sizes. The second is the latency overhead introduced by adapter loading and offloading. To tackle these challenges efficiently, we propose Unfied Paging and overlap the I/O with computation by prefetching adapter weights.

Understanding the nature of adapter weights is essential for optimizing memory usage. Our primary observation is that these dynamic adapter weights are analogous to dynamic KV caches in several ways:

- **Variable sizes and operations:** Just as the size of KV cache size fluctuates with the sequence length, the ranks of the active adapters can also depend on the choice of adapter associated with each request. KV caches are allocated when requests arrive and deallocated once the requests are completed. Similarly, adapter weights are loaded and cleared with each request. If not managed properly, this variability can result in fragmentation.
- **Dimensionality:** A KV cache tensor for a request in a layer has a shape of (S, H), where S denotes the sequence length and H represents the hidden dimension. Meanwhile, the shape of a LoRA weight is (R, H), with R standing for the rank and H the hidden dimension. Both share a dimension size of H that can be leveraged to reduce fragmentation.

Motivated by these parallels, we extend the idea of PagedAttention (Kwon et al., 2023) to Unified Paging which

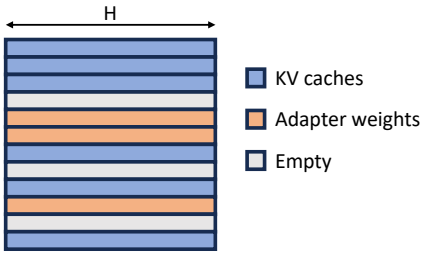

Figure 5. Unified memory pool. We use a unified memory pool to store both KV caches and adapter weights in a non-contiguous way to reduce memory fragmentation. The page size is H elements.

manages adapter weights in addition to the KV cache. Unified Paging uses a unified memory pool to jointly manage both KV cache and adapter weights. To implement this, we first allocate a large buffer statically for the memory pool. This buffer uses all available space except for the space occupied by the base model weights and temporary activation tensors. Both KV caches and adapter weights are stored in this memory pool in a paged manner, with each page corresponding to a vector of H. Thus, a KV cache tensor with a sequence length of S uses up S pages, while a LoRA weight tensor of rank R takes up R pages. Figure 5 illustrates the layout of our memory pool, where KV caches and adapter weights are stored interleaved and non-contiguously. This approach significantly reduces fragmentation, ensuring that adapters weights of various ranks can coexist with dynamic KV caches in a structured and systematic manner.

5 Evaluation of Offline Context Learning

Dataset We evaluate our system on the QuALITY (Pang et al., 2022) dataset. QuALITY is a multiple-choice question answering dataset for long contexts. It contains 150 articles with an average length of 5000 tokens, with 6737 questions in total. This dataset is particularly suited to evaluate models in a long-context setting.

Model configuration We use LLaMA-7B-chat (Touvron et al., 2023a) as our base model, which has a maximum context window of 4096 tokens. Our LoRA rank is set to 8 unless otherwise specified.

5.1 Setup

We experiment with using pretrained encoder from ICAE (Ge et al., 2023) as our context encoder. This encoder is a LLaMA2-7B-chat (Touvron et al., 2023a) model pretrained on the Pile (Gao et al., 2020) dataset and instruction fine-tuned on the self-curated PwC (Prompt-with-Context) dataset to perform general context compression task. The encoder takes a context as long as 4096 tokens as input, and can compress it to 128 memory tokens. Even though the maximum supported length is 4096 tokens, experiment results from ICAE demonstrates that its context compressor maintain the same performance with memory tokens when the original context length is below 512 tokens. In our setting, the context length is 4096 tokens. We will truncate the article to 4096 tokens if the article's length exceeds the context window limit.

In order to evaluate our methods performance on QuALITY , we consider the following scenarios.

1. **Base LLM with Question.** In this setting, we do not provide the context at all and only provide the multiple-choice questions to the base LLM. We use the pretrained LLaMA2-7B-chat as our model and do not perform additional training.

2. **Base LLM with Question + Context**. In this setting, we additionally prepend the article relevant to the question, and ask the LLM to answer the question. The context is truncated so that the total token length does not exceed 4096. This represents a baseline usage of LLMs on this task.

3. **Finetuned LLM (Next Token Prediction) with Question**. In this setting, we consider a naive finetuning method where we simply use LoRA to fine-tune on all the articles in the dataset using standard next-token prediction objective. During inference, we provide no context and directly append the question to the LLM.

4. **Off-the-shelf ICAE encoder with Question** In this setting, we directly use the pre-trained ICAE encoder to compress the context into memory tokens and prepend these tokens before the question. We do not perform any finetuning of our model. The LLM is supposed to answer the question conditioned on the memory tokens and the question.

5. **Finetuned LLM (ours) + Question**. In this setting, we use the offline context learning method proposed in our work to fine-tune the LLM. Since our context encoder is also a LLaMA-7B model, we finetune both the context encoder and the LLM using two separate LoRAs. During inference, we use the finetuned model, and use the ICAE context encoder to compress the context into 128 memory tokens, and prepend it before the question.

Our results are summarized in the following table.

Examining the results from Table 1, we observe a notable improvement in accuracy, increasing from 31% to 37%, when context is provided to the base LLM. However, simply finetuning the LLM with next token prediction on these

Base LLM w. Question	Base LLM w. Question + Context	Finetuned LLM (N.T.P) w. Question	Pretrained ICAE w. Question	Finetuned LLM (ours) w. Question
31%	37%	31%	27%	**39%**

Table 1. Multiple-Choice QA Accuracy on the QuALITY dataset

contexts does not yield effective results. For instance, finetuning the LLM without context during inference results in an unchanged accuracy of 31%, mirroring the base model's performance in a similar setting. Moreover, utilizing the off-the-shelf ICAE as a context encoder, without any additional finetuning, leads to a reduced accuracy of only 27%. This is even lower than the base LLM's performance without context, suggesting that the pretrained context encoder on its own is insufficient for learning the context information effectively.

In contrast, the application of our offline context learning method significantly enhances the model's performance, achieving an accuracy of 39%. This result not only surpasses the base model with context but also exceeds the in-context learning accuracy of 37%.

6 Evaluation of S-LoRA

We evaluate the performance of S-LoRA on both synthetic and real production workloads. S-LoRA is built on top of LightLLM (ModelTC, 2023), a single-model LLM serving system based on PyTorch (Paszke et al., 2019) and Triton (Tillet et al., 2019). We evaluate the scalability of S-LoRA by serving up to two thousand LoRA adapters simultaneously and compare it with other strong baselines. We then perform ablation studies to verify the effectiveness of individual components.

6.1 Setup

Model. We test the Llama model series (Touvron et al., 2023a;b), one of the most popular open large language models. We consider 3 different model and adapter configurations, which are listed in Table 2 Our optimizations can be easily adapted to other transformer-based architectures as well, such as GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2022; Anil et al., 2023).

Setting	Base model	Hidden size	Adapter ranks
S1	Llama-7B	4096	{8}
S2	Llama-7B	4096	{64, 32, 16, 8}
S4	Llama-13B	5120	{64, 32, 16}

Table 2. Model and adapter configurations.

Hardware. We conduct tests on various hardware settings, including a single NVIDIA A10G GPU (24GB), a single A100 GPU (40GB), a single A100 GPU (80GB), and multiple A100 GPUs (40GB/80GB). The host's main memory varies based on the GPU setup, ranging from 64 GB to 670 GB. We will show that S-LoRA can efficiently scale the number of adapters, limited only by the available main memory.

Baselines. We benchmark several variants of S-LoRA, HuggingFace PEFT (Mangrulkar et al., 2022), and vLLM (Kwon et al., 2023).

- "HuggingFace PEFT" is a library for training and running parameter-efficient fine-tuning models. It lacks advanced batching and memory management. We build a server using it that batches single adapter requests and switches adapter weights between batches.
- "vLLM m-packed" is a simple multi-model serving solution based on vLLM, a high-throughput serving system. Because vLLM does not support LoRA, we merge the LoRA weights into the base model and serve the multiple versions of the merged weights separately. To serve m LoRA adapters, we run m vLLM workers on a single GPU, where multiple workers are separate processes managed by NVIDIA MPS. We statistically allocate the GPU memory proportionally to the average request rate for each process.
- "S-LoRA" is S-LoRA with all the optimizations and it is using the first-come-first-serve scheduling strategy.
- "S-LoRA-no-unify-mem" is S-LoRA without the unified memory management.
- "S-LoRA-bmm" is S-LoRA without unified memory management and customized kernels. It copies the adapter weights to continuous memory space and performs batched matrix multiplication with padding.

Metrics. There are several metrics to measure the performance of serving systems, including latency and throughput. Following common practice, we report the *throughput, average request latency, average first token latency, and SLO attainment*. SLO attainment is defined as the percentage of requests that return the first token in 6 seconds. Additionally, we introduce a new metric termed *user satisfaction*, which offers a more fine-grained analysis of the first token latency. Intuitively, a shorter first token latency gives a higher satisfaction. The satisfaction becomes 0 if the first token latency exceeds the SLO.

6.2 End-to-End Results on Synthetic Workloads

Workload trace. We generate synthetic workload traces using the Gamma process. Given n adapters, the requests for adapter i are modeled using a Gamma arrival process with a mean rate of λ_i and a coefficient of variance (CV) of cv. The mean rate, λ_i, adheres to a power-law distribution with an exponent α. The total request rate for all adapters is R requests per second. For the n adapters, we set their ranks based on the list provided in Table 2 with a round-robin method. Our tests cover various combinations of n, α, R, and cv. For every request, the input and output lengths are sampled from uniform distributions $U[I_l, I_u]$ and $U[O_l, O_u]$ respectively. The default duration of a trace is 5 minutes. To conduct comprehensive experiments, we first pick a set of default parameters for generating workloads, as shown in Table 3. We then vary one of the *n, α, R, and cv* to see how each factor affects the performance.

Table 3. Default parameters for generating the synthetic workloads. "7B @ A10G" means running a Llama-7B on a single A10G.

Setting	n	α	R	cv	$[I_l, I_u]$	$[O_l, O_u]$
7B @ A10G (24G)	200	1	2	1	[8, 512]	[8, 512]
7B @ A100 (80G)	200	1	10	1	[8, 512]	[8, 512]
13B @ A100 (40G)	200	1	2	1	[8, 512]	[8, 512]
13B @ A100 (80G)	400	1	6	1	[8, 512]	[8, 512]

Table 4. Throughput (req/s) comparison between S-LoRA, vLLM-packed, and PEFT. The hardware is a single A100 (80GB). We run PEFT for a shorter duration when $n = 100$. We do not evaluate PEFT for $n \geq 1000$, as its throughput is already very low for a small n. "OOM" denotes out-of-memory.

Model Setup	n	S-LoRA	vLLM-packed	PEFT
S1	5	8.05	2.04	0.88
	100	7.99	OOM	0.25
	1000	7.64	OOM	-
	2000	7.61	OOM	-
S2	5	7.48	2.04	0.74
	100	7.29	OOM	0.24
	1000	6.69	OOM	-
	2000	6.71	OOM	-
S4	2	4.49	3.83	0.54
	100	4.28	OOM	0.13
	1000	3.96	OOM	-

Comparison with other systems. We compare S-LoRA with both vLLM-packed and HuggingFace PEFT for serving many LoRA adapters. The results are shown in Table 4. Remarkably, S-LoRA can serve 2,000 adapters simultaneously, maintaining minimal overhead for the added LoRA computation. In contrast, vLLM-packed needs to maintain multiple weight copies and can only serve fewer than 5 adapters due to the GPU memory constraint. The throughput of vLLM-packed is also much lower due to the missed batching opportunity. Although PEFT can swap adapters between batches, enabling it to handle a large number of adapters, its lack of advanced batching methods and memory management results in significantly worse performance. Overall, S-LoRA achieves a throughput up to 4x higher than vLLM-packed when serving a small number of adapters, and up to 30x higher than PEFT, while supporting a significantly larger number of adapters.

Comparing with own variants. Since no baseline system can efficiently scale to a large number of adapters, we now focus on comparing S-LoRA with its own variants. Figure 6 illustrates how they scale with the number of adapters. S-LoRA achieves noticeably higher throughput and lower latency compared to S-LoRA-bmm and S-LoRA-no-unify-mem. This implies that our memory pool and custom kernels are effective. When the number of adapters increases, the throughput of S-LoRA initially experiences a slight decline due to the overhead introduced by LoRA. However, once the number of adapters reaches a certain threshold (e.g., 100 in most experiments), the throughput of S-LoRA no longer decreases. This stability can be attributed to the fact that as the number of adapters grows, the number of activated adapters for the currently running batch remains unchanged, maintaining a constant overhead. Consequently, S-LoRA can scale to a much larger number of adapters without incurring additional overhead, constrained only by the available main memory.

6.3 Ablation Study

Merging adapter weights versus computing on-the-fly. While S-LoRA does not merge adapter weights and computes LoRA matrices on-the-fly each time, we compare it with an alternative design that merges an adapter with the base model, denoted as $x(W + AB)$, as proposed in the LoRA paper. This approach involves: 1) Updating the base model with the current adapter weights before each new batch; and 2) Switching to a new adapter if there are too many waiting requests.[2] This method is efficient for a small number of adapters due to the reduced LoRA computation overhead.

Results in Figure 7 demonstrate that with just one adapter, the merging approach outperforms the on-the-fly computation owing to a one-time merging cost. However, its performance declines with more than 2 adapters, primarily because of the time-consuming switch between adapters. Such switching results in periods of GPU under-utilization. Furthermore, a smaller value of α causes requests to be distributed unevenly across adapters, which in turn reduces batch sizes and overall performance.

[2]This is different from PEFT. For example, it has continuous batching and PagedAttention, which are not enabled in PEFT.

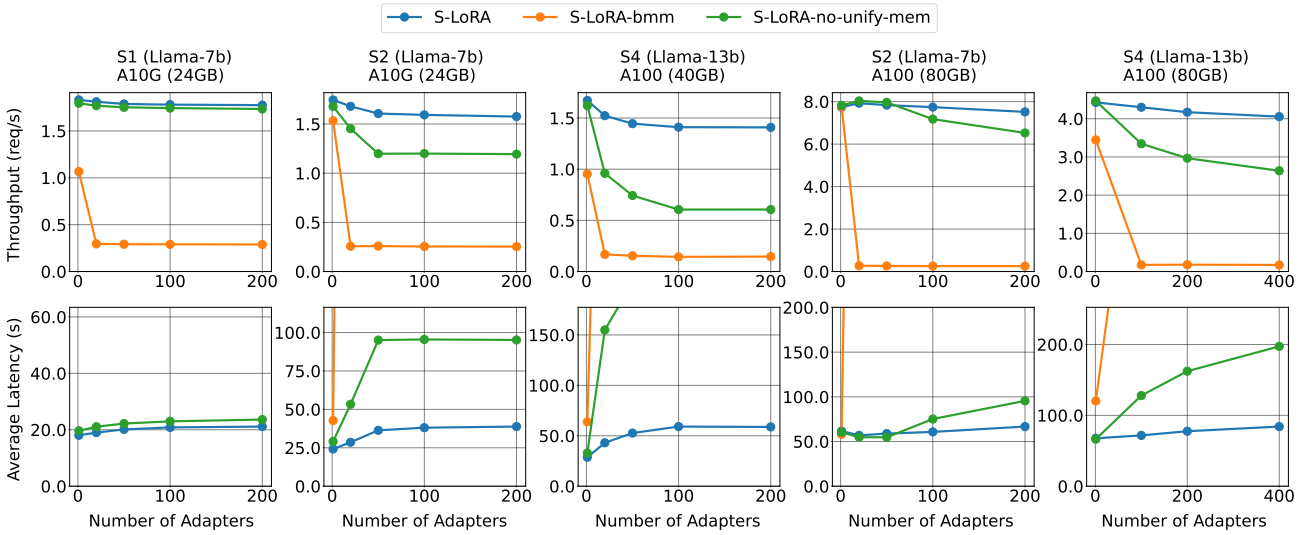

Figure 6. The throughput and average request latency of S-LoRA and its variants under different numbers of adapters. S-LoRA achieves significantly better performance and can scale to a large number of adapters. We run S-LoRA-bmm for a shorter duration since it has a significantly lower throughput. Some S-LoRA-bmm curves are omitted because it is out of the figures's scope.

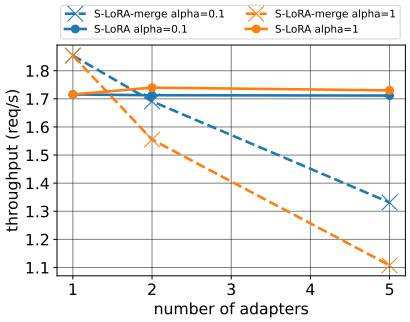

Figure 7. Ablation study comparing adapter merging and on-the-fly compute for S2 on A10G (24GB) with different α and number of adapters. The settings for the synthetic workloads are $R = 2$, $cv = 1$, $[I_l, I_u] = [8, 512]$, $[O_l, O_u] = [8, 512]$.

7 Related Work

Long-context LLMs. Pretrained LLMs have a fixed-size context window, which usually has a length of 4096 as in LLaMA 2 (Touvron et al., 2023a) or 8192 as in GPT-4 (versions 0314 and 0613). Further increasing the context window sizes during pretraining has been challenging for a long time due to the high cost induced by the quadratic time and memory complexities of exact attention. Recently, there have been many attempts to extend the context window of LLMs with continued training or fine-tuning. Positional interpolation (Chen et al., 2023b) extends the context lengths of LLaMA family models to 32k by finetuning with scaled rotary position encodings, which applies to RoPE-based pretrained LLMs (Su et al., 2021). LLaMA 2 Long (Xiong et al., 2023) applies a similar technique but with continual pretraining. Mistral (Jiang et al., 2023a) proposes sliding window attention that only allows the token to attend to a certain number of tokens from the previous layer, reducing compute costs and enabling pretraining with long-context to 32k. Nevertheless, as the auto-regressive generation in LLMs is largely memory-bound, storing the KV cache of such extended long contexts slows down the inference and creates requirements for large GPU VRAMs.

Context compression. A closely related but different area of research is context compression. In this area, the goal is to train a general compressor that can compress any input prompts to a shorter one. GIST (Mu et al., 2023), Auto-Compressor (Chevalier et al., 2023), and ICAE (Ge et al., 2023) fine-tune LLMs in a "soft prompt-tuning" manner, either applying specific regularization in attention masks or utilizing dedicated "memory tokens" to compress contexts into embeddings with significant shorter lengths. LLM-Lingua (Jiang et al., 2023c;b) proposes a question-aware coarse-to-fine compression framework to compress prompts in black-box LLM APIs. Another line of work employs KV cache eviction (Zhang et al., 2023c; Liu et al., 2023c), which only caches a subset of keys and values (denoted as "Heavy Hitters" or "pivotal tokens"), and removes tokens that are uninformative for future outputs during the inference. Those methods reduce the memory footprint and essentially achieve the similar goal of prompt compression. However, all those approaches aim to compress any inputs seen during the inference time, which results in rapid performance drops as the compression ratio exceeds a certain limit (e.g. 25-50%) and they usually incur considerable performance drops for out-of-distribution texts. In this work, we mainly focus on extreme compression of in-distribution documents of interest, going down to 32× compression

rates.

Optimize LLM serving with system techniques. The significance of the transformer architecture has led to the development of many specialized serving systems for it. These systems use advanced batching mechanisms (Fang et al., 2021; Yu et al., 2022), memory optimizations (Sheng et al., 2023; Kwon et al., 2023), GPU kernel optimizations (Wang et al., 2021; Aminabadi et al., 2022; NVIDIA, 2023; Dao, 2023), model parallelism (Pope et al., 2022; Aminabadi et al., 2022), parameter sharing (Zhou et al., 2022), and speculative execution (Stern et al., 2018; Miao et al., 2023) for efficient serving. Among them, PetS (Zhou et al., 2022) is most relevant to ours. However, PetS only considers the serving for small encoder-only BERT models. It does not consider generative inference, a very large number of adapters or large models go beyond a single GPU, so it does not address the problems in our settings.

In a concurrent work (Chen et al., 2023a), the concept of decomposed computation for the base model and adapters was explored. Our CUDA kernels were developed based on the implementation presented in a prior blog post of this study, with additional support for batching different ranks and non-contiguous memory. Moreover, our novel memory management and tensor parallelism techniques have not been covered in any previous work.

Optimize LLM serving with algorithm techniques. In addition to system-level improvements, inference efficiency can be enhanced using algorithm techniques like quantization (Yao et al., 2022; Dettmers et al., 2022; Frantar et al., 2022; Xiao et al., 2023; Lin et al., 2023), sparsification (Frantar & Alistarh, 2023; Zhang et al., 2023b) and model architecture improvements (Shazeer, 2019). These approaches can reduce memory consumption and accelerate the computation, with a minor compromise in model quality. They are complementary to the techniques in this paper.

Parameter-efficient fine-tuning. Recent work has developed methods for parameter-efficient fine-tuning of large pre-trained language models. These methods show fine-tuning is possible with only a small fraction of tuned parameters. The state-of-the-art methods include LoRA (Hu et al., 2021), Prefix-tuning (Li & Liang, 2021), P-Tuning (Liu et al., 2021), Prompt tuning (Liu et al., 2023b; Lester et al., 2021), AdaLoRA (Zhang et al., 2022), and $(IA)^3$ (Liu et al., 2022). While our paper focuses on LoRA due to its wide adoption, most techniques can be easily applied to other parameter-efficient fine-tuning methods as well.

General purpose model serving systems. Over the years, the domain of general model serving has seen significant advancements, Notable systems from earlier research include Clipper (Crankshaw et al., 2017), TensorFlow Serving (Olston et al., 2017), Nexus (Shen et al., 2019), InferLine (Crankshaw et al., 2020), and Clockwork (Gujarati et al., 2020). These systems delve into topics such as batching, caching, and model placement, catering to both individual and multiple model deployments. In more recent developments, DVABatch (Cui et al., 2022), REEF (Han et al., 2022), Shepherd (Zhang et al., 2023a) and AlpaServe (Li et al., 2023) have explored the ideas of multi-entry multi-exit batching, preemption, and statistical multiplexing with model parallelism. Although these systems have made significant contributions, they overlook the auto-regressive characteristics and parameter-efficient adapters in LLM serving, leading to potential optimization gaps.

8 Conclusion

In this work, we propose offline context learning, a new paradigm to address the long-context challenge by pre-processing the long context prior to inference through parameter-efficient finetuning. Our preliminary results on the QuALITY dataset show that our offline learning method outperforms in-context learning while processing over 32x fewer tokens during inference.

To serve the workload of offline context learning, we propose S-LoRA, a serving system scalable to thousands of LoRA adaptors on a single GPU. Compared to state-of-the-art libraries such as HuggingFace PEFT and vLLM (with naive support of LoRA serving), our serving system S-LoRA can improve the throughput by up to 4 times and increase the number of served adapters by several orders of magnitude.

References

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., and He, Y. Deepspeed- inference: Enabling efficient inference of transformer models at unprecedented scale. In Wolf, F., Shende, S., Culhane, C., Alam, S. R., and Jagode, H. (eds.), *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022.

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, L., Ye, Z., Wu, Y., Zhuo, D., Ceze, L., and Krishnamurthy, A. Punica: Multi-tenant lora serving, 2023a.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv: 2306.15595*, 2023b.

Chevalier, A., Wettig, A., Ajith, A., and Chen, D. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 613–627, 2017.

Crankshaw, D., Sela, G.-E., Mo, X., Zumar, C., Stoica, I., Gonzalez, J., and Tumanov, A. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 477–491, 2020.

Cui, W., Zhao, H., Chen, Q., Wei, H., Li, Z., Zeng, D., Li, C., and Guo, M. Dvabatch: Diversity-aware multi-entry multi-exit batching for efficient processing of dnn services on gpus. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 183–198, 2022.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., Li, L., and Sui, Z. A Survey on In-context Learning, June 2023. URL http://arxiv.org/abs/2301.00234. arXiv:2301.00234 [cs].

Fang, J., Yu, Y., Zhao, C., and Zhou, J. Turbotransformers: an efficient gpu serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 389–402, 2021.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Frantar, E. and Alistarh, D. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Ge, T., Hu, J., Wang, L., Wang, X., Chen, S.-Q., and Wei, F. In-context Autoencoder for Context Compression in a Large Language Model, October 2023. URL http://arxiv.org/abs/2307.06945. arXiv:2307.06945 [cs].

Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., and Mace, J. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462, 2020.

Han, M., Zhang, H., Chen, R., and Chen, H. Microsecond-scale preemption for concurrent {GPU-accelerated}{DNN} inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 539–558, 2022.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a.

Jiang, H., Wu, Q., , Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *ArXiv preprint*, abs/2310.06839, 2023b. URL https://arxiv.org/abs/2310.06839.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, December 2023c. URL https://arxiv.org/abs/2310.05736.

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

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 Flinn, J., Seltzer, M. I., Druschel, P., Kaufmann, A., and Mace, J. (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023*, pp. 611–626. ACM, 2023.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, 2021.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. A. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35: 1950–1965, 2022.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the Middle: How Language Models Use Long Contexts, November 2023a. URL http://arxiv.org/abs/2307.03172. arXiv:2307.03172 [cs].

Liu, X., Ji, K., Fu, Y., Tam, W. L., Du, Z., Yang, Z., and Tang, J. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., and Tang, J. Gpt understands, too. *AI Open*, 2023b. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2023.08.012. URL https://www.sciencedirect.com/science/article/pii/S2666651023000141.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023c.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., and Bossan, B. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Wong, R. Y. Y., Chen, Z., Arfeen, D., Abhyankar, R., and Jia, Z. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

ModelTC. Lightllm: Python-based llm inference and serving framework. https://github.com/ModelTC/lightllm, 2023. GitHub repository.

Mu, J., Li, X. L., and Goodman, N. Learning to Compress Prompts with Gist Tokens, July 2023. URL http://arxiv.org/abs/2304.08467. arXiv:2304.08467 [cs].

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Ni, J., Qu, C., Lu, J., Dai, Z., Ábrego, G. H., Ma, J., Zhao, V. Y., Luan, Y., Hall, K. B., Chang, M.-W., and Yang, Y. Large dual encoders are generalizable retrievers, 2021.

NVIDIA. Fastertransformer. https://github.com/NVIDIA/FasterTransformer, 2023.

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., and Soyke, J. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

Pang, R. Y., Parrish, A., Joshi, N., Nangia, N., Phang, J., Chen, A., Padmakumar, V., Ma, J., Thompson, J., He, H., and Bowman, S. R. Quality: Question answering with long input texts, yes!, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *arXiv preprint arXiv:2211.05102*, 2022.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Shen, H., Chen, L., Jin, Y., Zhao, L., Kong, B., Philipose, M., Krishnamurthy, A., and Sundaram, R. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 322–337, 2019.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning, ICML 2023*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31094–31116. PMLR, 2023.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Stern, M., Shazeer, N., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *NEUROCOMPUTING*, 2021. doi: 10.1016/j.neucom.2023.127063.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, X., Xiong, Y., Wei, Y., Wang, M., and Li, L. Lightseq: A high performance inference library for transformers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pp. 113–120, 2021.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023.

Xiong, W., Liu, J., Molybog, I., Zhang, H., Bhargava, P., Hou, R., Martin, L., Rungta, R., Sankararaman, K. A., Oguz, B., Khabsa, M., Fang, H., Mehdad, Y., Narang, S., Malik, K., Fan, A., Bhosale, S., Edunov, S., Lewis, M., Wang, S., and Ma, H. Effective long-context scaling of foundation models. *arXiv preprint arXiv: 2309.16039*, 2023.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

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, 2022.

Zhang, H., Tang, Y., Khandelwal, A., and Stoica, I. SHEPHERD: Serving DNNs in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 787–808, Boston, MA, April 2023a. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/zhang-hong.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2022.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023b.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv: 2306.14048*, 2023c.

Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E. P., et al. Alpa: Automating inter-and intra-operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 559–578, 2022.

Zhou, Z., Wei, X., Zhang, J., and Sun, G. {PetS}: A unified framework for {Parameter-Efficient} transformers serving. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 489–504, 2022.