# PQCache: Product Quantization-based KVCache for Long Context LLM Inference

Hailin Zhang
Peking University

Xiaodong Ji
Peking University

Yilin Chen
Beijing Institute of Technology

Fangcheng Fu
Peking University

Xupeng Miao
Carnegie Mellon University

Xiaonan Nie
Peking University

Weipeng Chen
Baichuan Inc.

Bin Cui
Peking University

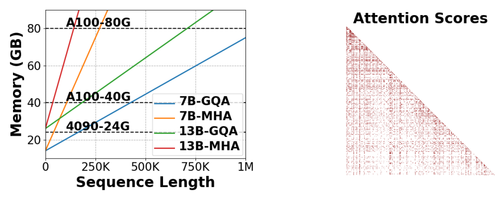
## ABSTRACT

As the field of Large Language Models (LLMs) continues to evolve, the context length in inference is steadily growing. Key-Value Cache (KVCache), a crucial component in LLM inference, has now become the primary memory bottleneck due to limited GPU memory. Current methods selectively determine suitable keys and values for self-attention computation in LLMs to address the issue. However, they either fall short in maintaining model quality or result in high serving latency. Drawing inspiration from advanced embedding retrieval techniques used in the database community, we consider the storage and searching of KVCache as a typical embedding retrieval problem. We propose **PQCache**, which employs Product Quantization (PQ) to manage KVCache, maintaining model quality while ensuring low serving latency. During the prefilling phase, we apply PQ to tokens' keys for each LLM layer and head. During the autoregressive decoding phase, for each newly generated token, we first identify important tokens through Maximum Inner-Product Search (MIPS) using PQ codes and centroids, then fetch the corresponding key-value pairs for self-attention computation. Through meticulous design of overlapping and caching, we minimize any additional computation and communication overhead during both phases. Extensive experiments show that PQCache achieves both effectiveness and efficiency. It maintains model quality even when only 1/5 of the tokens are involved in attention, while attaining acceptable system latency.

## 1 INTRODUCTION

With the emergence of ChatGPT [40], Large Language Models (LLMs) have captured the attention of researchers and engineers as promising candidates for Artificial General Intelligence (AGI). LLMs exhibit exceptional performance in the "next token prediction" task, where they take a sequence of tokens as input (also called prompt) and generate subsequent tokens autoregressively during inference. Constructed with transformer layers, the fundamental mechanism of LLMs is the self-attention module. For each token, this module computes "query", "key", and "value" representations. Each token's query interacts with the previous tokens' keys (including itself) to derive attention weights, which are then used for weighted summation of the previous tokens' values. Figure 2 illustrates a typical self-attention module within a transformer layer.

To accommodate increasingly lengthy prompts, the maximum input length of LLMs has expanded significantly, from 2K-4K [50, 52] to 32K [25, 51], 128K [15, 40], or even millions of tokens [2, 9, 31]. As illustrated in Figure 2, the process of LLM inference involves two phases: *prefilling* and *decoding*. During prefilling, LLMs handle the lengthy input and compute keys and values for all input tokens. During decoding, LLMs generate the next new token and produce its key and value. To avoid redundant computations, the keys and values of preceding tokens are commonly cached in the Key-Value Cache (KVCache), and fetched for subsequent tokens' attention computation. However, as prompts grow in length, the memory consumption of KVCache has far exceeded the memory capacity of each individual GPU, even for 7B and 13B LLMs in Figure 1(a). This poses a formidable challenge for modern LLM inference.



(a) Model and KVCache memory. (b) Attention example.

**Figure 1: Observations. The left figure shows the KVCache memory consumption of LLMs (for the meaning of MHA and GQA, please refer to Section 2.1). The right figure shows an example of attention scores on the MultiNews dataset [12], with darker colors indicating higher scores.**

Recognizing that specific tokens significantly influence generation, i.e. their attention weights are much larger than others [33, 63], numerous methods selectively incorporate these tokens within attention mechanisms while excluding others. This approach aims to address the memory challenge posed by KVCache and is commonly referred to as *selective attention* [37]. Related methods can be classified into two categories: KVCache dropping [33, 58, 63] and KVCache offloading [46, 57]. However, these methods either rely on improper assumptions or introduce notable latency during inference, failing to obtain both effectiveness and efficiency. KVCache dropping methods discard unnecessary key-value pairs, based on the assumption that unimportant tokens have no relevance for subsequent generation. Nevertheless, as shown in an attention score example in Figure 1(b), many tokens with lower average attention weights can still contribute to later generated tokens. Prior research [10, 29] also highlights the drawback of direct dropping. KVCache offloading methods, including InfLLM [57] and SPARQ [46], store the KVCache on CPU, and fetch relevant key-value pairs for each newly generated token according to easy-to-compute proxy scores. InfLLM organizes the KVCache into blocks,

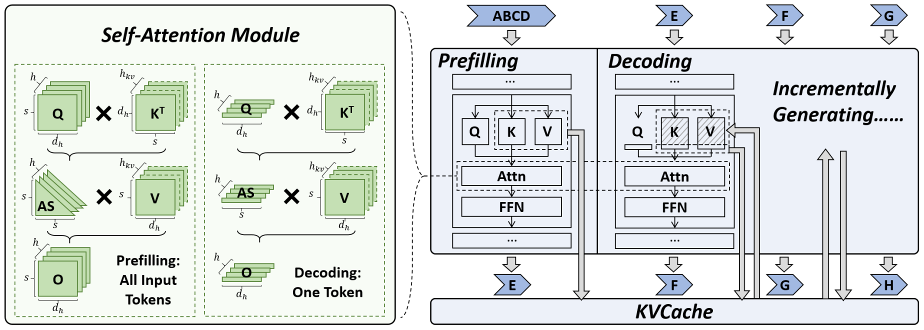


**Figure 2: An overview of LLM inference. The left part illustrates the computation process of the self-attention module, where "Q", "K", "V", "AS", and "O" represent query, key, value, attention score, and output, respectively. The right part depicts the LLM inference process, consisting of the prefilling phase and the decoding phase, where "Attn" and "FFN" represent the attention layer and the feed-forward network layer, respectively. The mathematical symbols are detailed in Table 1.**

using representative tokens within each block to compute relevance. Unfortunately, as shown in Figure 1(b), we do not observe the space-continuity assumption in InfLLM. SPARQ identifies a subset of dimensions with large magnitude in queries, and fetches only these dimensions from all keys to determine the most relevant tokens. Despite demonstrating effectiveness using a large number of dimensions, it incurs excessive communication overhead, and the serialized computation-communication process hinders opportunities for system optimization (e.g., prefetching). In summary, existing methods fall short in achieving both effectiveness and efficiency for long-context LLM inference.

We clarify that, selective attention computation, requiring to find the top-$k$ relevant tokens' key-value pairs according to query-key multiplications, is essentially a Maximum Inner-Product Search (MIPS) problem in the context of embedding retrieval. Embedding retrieval, a common research focus within the database and data management domains, encompasses many well-known methods including Product Quantization [18, 24], inverted index [6, 7], and graph-based methods [23, 35]. These methods mainly involve two operations: (1) index construction, where candidate embeddings are organized into a well-structured index; (2) searching, which aims to efficiently retrieve the top-$k$ relevant embeddings from the index for a given query embedding. Surprisingly, we found that the selective attention calculation process during LLM inference can be mapped into these operations. Specifically, the prefilling phase generates most of the KVCache and constructs the index, while the decoding phase finds relevant keys/values and updates KVCache using the newly generated token.

Inspired by the advanced embedding retrieval techniques, in this paper we propose **PQCache** to ensure both effectiveness and efficiency during long-context LLM inference. In the prefilling phase, we generate the KVCache, store it in CPU memory, and then construct index structures. In the decoding phase, we efficiently retrieve relevant key-value pairs for self-attention computation and update KVCache. Considering the latency requirements of LLM inference, we cannot employ methods with expensive index construction overheads, such as graph-based methods or complex inverted-index methods. We take the advantage of low-cost Product Quantization (PQ) [24] from embedding retrieval [23], where embeddings are initially partitioned into sub-embeddings and then clustered. The key idea of PQCache is to construct PQ codebooks using preceding token keys and perform MIPS to retrieve relevant key-value pairs for subsequent self-attention computations. We propose a system-algorithm co-design method based on PQ, leveraging both its high recall potentiality, and the opportunities for system optimization. Further experimental analysis show that PQCache improves the LongBench scores up to 6.21 compared to existing methods, and attains acceptable system latency.

To the best of our knowledge, this is the pioneering work that incorporates embedding retrieval technique to address the KVCache memory challenge. PQ offers a well-behaved approximation of embedding vectors (and their inner product), while consuming only a small amount of memory. In the prefilling phase, we apply PQ to the generated keys for each layer and head, and obtain PQ codes and centroids through clustering on CPU. At each autoregressive decoding step, we perform inner product between the partitioned query and the PQ centroids, then combine with PQ codes to obtain the approximate attention weights. Using the approximation, we retrieve top-$k$ relevant key-value pairs from CPU memory for self-attention computation, rather than accessing the entire KVCache.

To enable efficient LLM inference, we carefully design the PQCache system to reduce latency. We implement prefetching and overlapping as much as possible: KVCache offloading, PQ construction, and the fetching of PQ codes and centroids are overlapped with LLM computation. To maximize the utilization of available GPU memory and minimize CPU-GPU communication, we additionally introduce a block-level cache on GPU, specifically for frequently accessed key-value pairs.

We summarize our contributions as follows:

- We incorporate the embedding retrieval technique PQ into KVCache management to enable both effective and efficient LLM inference.
- We propose a system-algorithm co-designed approach PQCache to approximately retrieve the top-$k$ relevant keys for a given query, with meticulous design of overlapping and caching.

- We evaluate PQCache through extensive experiments. It maintains model quality with 1/5 of the tokens in attention, while achieving acceptable system latency.

**Table 1: Notations. "#" means "the number of".**

| Sym. | Explanation | Sym. | Explanation |
|---|---|---|---|
| $n$ | Batch size. | $m$ | # partitions in PQ. |
| $s$ | Current sequence length. | $b$ | # bits for PQ codes. |
| $d$ | Hidden states dimension. | $d_m$ | Dimension of each partition. |
| $h_{(kv)}$ | # heads (for keys and values). | $k$ | # tokens in selective attention. |
| $d_h$ | Dimension of each head. | $T$ | # K-Means iterations. |

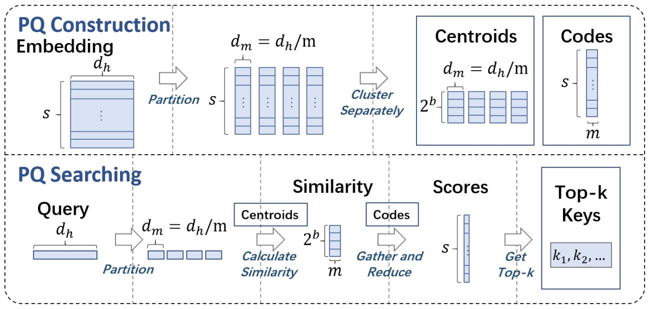


**Figure 3: An overview of PQ construction and searching.**

## 2 PRELIMINARY

In this section, we introduce fundamental concepts related to LLM, PQ, and the memory hierarchy.

### 2.1 Large Language Model Inference

An overview of LLM inference is depicted in Figure 2. An LLM comprises a stack of transformer layers, along with a vocabulary embedding for input and a token classifier for output. The self-attention module, which is a crucial component of a transformer layer, facilitates interaction and information aggregation among different tokens. Multi-Head Attention (MHA) and Grouped-Query Attention (GQA) [3] are the primary variants of the self-attention module. Following the notations in Table 1, the attention module receives an input of shape $(n, s, d)$. In MHA, the input is separately projected and transposed for query, key, value, resulting in the same shape of $(n, h, s, d_h)$, where it usually holds that $d = h * d_h$. Different heads are expected to capture different semantic information. The attention mechanism multiplies the queries and keys, applies a lower-triangular mask to restrict queries to preceding keys only, and performs softmax to obtain the attention scores of shape $(n, h, s, s)$. The attention scores are then used to weighted-sum the values, yielding an output of shape $(n, h, s, d_h)$, which is later reshaped into $(n, s, d)$. To alleviate memory and computation burden, GQA employs a smaller number of heads $h_{kv}$ for keys and values, resulting in their shape being $(n, h_{kv}, s, d_h)$. In this setup, each key-value pair corresponds to multiple queries.

During LLM inference, each execution of the model generates a new token, following an autoregressive manner. The first traversal and the subsequent traversals of the LLM are referred to as "prefilling" and "decoding" separately, as shown in Figure 2. During the prefilling phase, the self-attention module computes the queries, keys, and values for all input tokens, and stores the key-value pairs as KVCache for later usage. During the autoregressive decoding phase, the attention module only computes the query, key, value for the last generated token. It leverages previous keys and values from the KVCache, and computes an attention score of shape $(n, h, 1, s)$. Concurrently, the newly generated key and value are added to the KVCache. Consequently, the memory consumption of KVCache scales linearly with the sequence length, which leads to a memory bottleneck in scenarios involving long-context LLM inference.

### 2.2 Product Quantization

PQ [24] was proposed to facilitate efficient Approximate Nearest Neighbor Search (ANNS), retrieving relevant embeddings from a large pool of candidates given a query embedding. MIPS is a special case of ANNS that uses inner product as similarity. As shown in Figure 3, PQ divides each candidate embedding into $m$ partitions, essentially decomposing the original embedding space into $m$ separate sub-spaces. Each sub-space undergoes K-Means clustering to group the sub-embeddings, yielding $2^b$ centroids. Each embedding is assigned $m$ codes, each having $b$ bits, corresponding to the centroids. These compact PQ codes enable the reconstruction of approximate embeddings with reduced memory requirements. During ANNS, the query embedding computes similarity with the centroids and aggregates the similarity using PQ codes, bypassing the need for full similarity calculations with every embedding.

PQ has a profound impact on ANNS, with its principles integrated into various efficient ANNS methods [6, 23, 27]. PQ has several variants, including Optimized PQ [18], Residual Quantization [36], and SCaNN [19]. While PQ was initially designed for ANNS, its variants are also applied in various learning tasks [30, 53, 61] to achieve effective compression and efficient computation.

### 2.3 GPU-CPU Memory Hierarchy

Modern deep learning tasks heavily rely on GPUs for executing compute-intensive operations. The GPU-CPU structure forms a typical memory hierarchy: the more expensive GPU memory offers faster memory I/O speeds for computation, while the CPU memory, connected via PCIe or NVLink, provides lower bandwidth. As model parameters increase and the demand for intermediate results storage (such as KVCache) grows, CPUs are often employed to share the memory load. Numerous research studies in machine learning systems propose offloading certain model parameters or activations to the CPU memory [39, 42, 45, 47], thereby enhancing the overall performance of GPU-centric deep learning tasks. The primary challenge in this context is to effectively schedule memory I/O (or say GPU-CPU communication) in conjunction with GPU computation to efficiently hide the associated overhead.

## 3 PQCACHE

In this section, we introduce PQCache, a novel system-algorithm co-designed method to enable effective and efficient long context LLM inference with large-scale KVCache. Figure 4 provides an overview

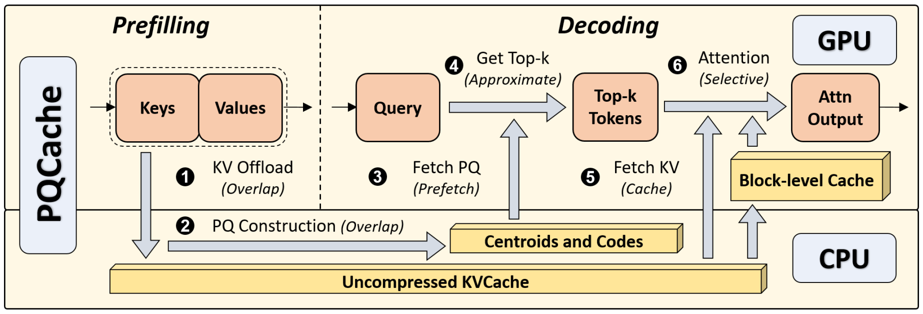
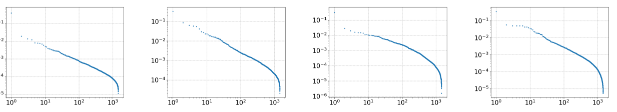


**Figure 4: An overview of PQCache. For simplicity, we only illustrate the process for a single transformer layer.**

of PQCache, where the KVCache from the prefilling phase is first offloaded to CPU, compressed using PQ, then fetched on demand through MIPS during the decoding phase.

## 3.1 Overview

We design PQCache to reserve all the KVCache in CPU, and selectively fetch relevant key-value pairs for self-attention computation. In long context inference scenario, the entire KVCache is too large for both attention computation and I/O communication within the memory hierarchy. Therefore, a common technique is to only perform attention on a subset of the key-value pairs, a process known as "selective attention". According to previous research [1, 17, 33, 44, 46, 55, 57, 59, 63], attention score is a proper metric to measure the importance or relevance of previous tokens. As shown in Figure 5, we plot the attention score distributions at several randomly-selected positions on an example from the XSUM dataset [38]. The attention scores generally follow powerlaw distributions, indicating that a small part of tokens are more important than most other tokens. Therefore, we can only include those tokens with large scores for self-attention computation. Following prior works [20, 58, 63], we also include initial tokens and the most recent tokens (called local tokens) in attention computation.



(a) Layer 3, head 25. (b) Layer 11, head 15. (c) Layer 20, head 27. (d) Layer 21, head 16.

**Figure 5: Distributions of attention scores.**

As detailed in Section 2.1, attention scores are calculated using a softmax function applied to the product of the current query and preceding keys. The procedure of identifying the top-$k$ keys with the highest scores fundamentally constitutes a Maximum Inner Product Search (MIPS) operation. Therefore, we try to leverage embedding retrieval techniques to enable effective selective attention and address the KVCache memory issue. Based on the observations above, we design PQCache, which offloads all the KVCache to CPU, and fetch only relevant tokens' key-values pairs during the decoding phase. Calculating exact attention scores of all previous tokens involves costly I/O communication, which is unacceptable in long context LLM inference. Inspired by Approximate Nearest Neighbor Search (ANNS) [6, 23, 27], we leverage the light-weight Product Quantization (PQ) method [24], which compress the vectors by partitioning and K-Means clustering. Though there are other ANNS methods (e.g. graph-based methods [13, 23, 35]) that can achieve better recall performance, they suffer from a computationally expensive construction process which may hinder LLM inference.

In PQCache, we construct PQ at the prefilling phase and utilize PQ at the decoding phase. At the prefilling phase, we need to calculate all the input tokens' keys and values for the self-attention module. After obtaining the keys, which have the shape of $(n, h_{kv}, s, d_h)$, we can construct PQ for each sample and each head. Concretely, for a tensor of shape $(s, d_h)$, we further split the dimension $d_h$ into $m$ sub-spaces with dimension $d_m$. For partitioned vectors $(m, s, d_m)$, where $d_m = d_h/m$, we conduct K-Means clustering separately for each group and get the centroids of shape $(m, 2^b, d_m)$ and PQ codes of shape $(s, m)$. Each PQ code, which indicates the cluster that the vector belongs to, only consumes $b$ bits to store.

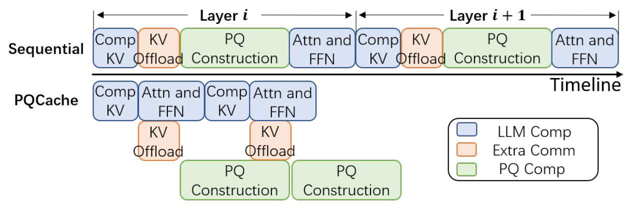
At the decoding phase, we first perform matrix multiplication between the query and the PQ centroids, then aggregate the results for all the tokens according to PQ codes. We can determine the top-$k$ relevant tokens using the approximate scores (before softmax) After fetching the approximate top-$k$ key-value pairs from CPU, the self-attention computation continues with retrieved tokens. Unlike normal embedding retrieval tasks, in LLM inference, newly generated keys and values are added into the KVCache. These tokens are first regarded as local tokens and reserved in GPU. When they are evicted from the sliding window of local tokens, they are assigned PQ codes based on their nearest centroids.

## 3.2 Complexity Analysis

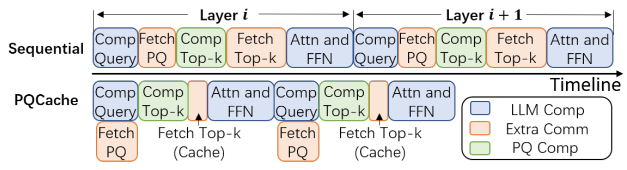
During the prefilling phase, we do not modify the attention computation, so the complexity remains the same for both time and memory. The additional K-Means clustering process has an average complexity of $O(s \cdot m \cdot 2^b \cdot T)$, where $T$ is the number of K-Means iterations. We leverage the idle CPU resources to perform K-Means, which is detailed in Section 3.3. During the decoding phase, the original attention time complextiy is $O(s \cdot d + d^2)$. In PQCache, we first conduct multiplication on PQ centroid with a time complexity of $O(s \cdot m + d^2)$, then compute the attention with a time complexity of $O(k \cdot d)$. The memory complexity is $O(s \cdot m + 2^b \cdot d)$, containing PQ centroids and PQ codes. Considering that $m \ll d$,

$k \ll s$, and $b$ is usually a value smaller than 10, the time complexity $O(s \cdot m + k \cdot d + d^2)$ and the memory complexity are much smaller than the original ones.

To facilitate efficient long context LLM inference, the design goal of PQCache is to provide overhead-agnostic service. Figure 6 illustrates the computation and communication involved in the PQCache-enabled LLM inference, covering both the prefilling and decoding phases. The original LLM computation is filled with blue color, while the computation and the communication introduced by PQCache are filled with green and red colors, which can be divided into four parts: (1) KVCache offloading and PQ structure fetching; (2) PQ construction using K-Means clustering; (3) Approximate top-$k$ computation; (4) The fetch process of top-$k$ relevant tokens' key-value pairs. As shown in Figure 6, except for the low-cost top-$k$ approximation, we employ distinct system design to eliminate these computation or communication overhead. The system design is detailed in the following sections.



(a) Prefilling.



(b) Decoding.

**Figure 6: PQCache v.s. sequential scheduling.**

## 3.3 Prefilling Phase

At the prefilling phase, on obtaining the input tokens' keys and values in each layer, they can attend to attention computation and be offloaded to CPU simultaneously. Given that the attention computation time scales quadratically with sequence length, while the communication time scales linearly, the communication can be fully overlapped in long context scenarios, as shown in Figure 7.

The K-Means clustering is of great complexity according to Section 3.2. To enable overhead-agnostic inference, we aim to fully utilize the idle CPU resources for clustering. However, as shown in Figure 7, the clustering process on CPU, including PQ construction for all heads in each layer, consumes more time than one-layer transformer computation on GPU at the prefilling stage. This is because the computational capability of GPUs has grown rapidly over the past decades, whereas CPUs are not specifically designed for computationally intensive tasks. To address the issue, we propose an adaptive K-Means clustering process which limits the number of

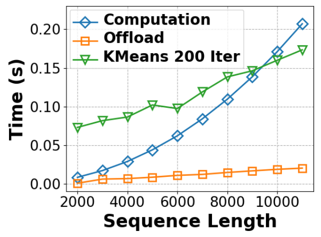


**Figure 7: The execution time of one-layer transformer computation, offloading, and clustering at the prefilling phase.**

iterations for clustering, ensuring that the clustering can be overlapped by GPU computation. For any given models and devices, we profile the computation time of one-layer transformer and K-Means clustering under different sequence lengths. By modeling the relationship between computation time and sequence length, we can determine the maximum number of K-Means iterations that can overlap with the computation for any given sequence length. In Section 4.3.3, we empirically study the trade-off between the efficiency and model quality of different clustering iterations.

## 3.4 Decoding Phase

At the decoding phase, the constructed PQ structure needs to be utilized by the attention module in each layer. While the preceding computation is underway, the PQ centroids and codes of the current layer can be pre-fetched in parallel. Since the PQ structure consumes negligible memory according to Section 3.2, its communication can directly overlap with decoding phase computation.

Throughout the entire inference, the only communication that cannot be overlapped is the retrieval of the top-$k$ relevant tokens, because it is dependent on preceding PQ approximation computation. Inspired by previous research [33, 57, 63], there are certain pivotal tokens that are consistently important during the inference process. Therefore, we maintain an GPU cache for these tokens. Following LM-Infinite [20] and StreamingLLM [58], we first preserve initial tokens and local tokens in the cache. For the remaining tokens, we maintain a block-level cache, wherein the cache structure resides on CPU and the storage is allocated on GPU. We construct token-blocks to reduce cache overhead, and cache the blocks on GPU. During each decoding step, we identify the blocks containing top-$k_{cache}$ hit tokens, and use them to fetch tokens while updating the cache structure. We employ asynchronous updates to avoid additional overhead. Experimental results in Section 4.3.4 illustrate the cache hit-rate, which helps reduce overall communication.

## 4 EXPERIMENTS

In this section, we conduct experiments and compare PQCache with existing methods. We experimentally show that PQCache achieves both effectiveness and efficiency.

## 4.1 Experimental Setup

*4.1.1* ***Models.*** We conduct experiments using two representative open-source LLMs: LLaMA-2-7B-Chat [52] and Mistral-7B-Instruct-v0.2 [25]. The former employs MHA and supports 4K context length, while the latter uses GQA and supports 32K context length. Both models share similar LLM architectures. We use FP16 for both models, which is a common practice in LLM inference.

4.1.2 ***Tasks.*** We evaluate PQCache on LongBench [5], a widely-used benchmark for long-context LLM inference. Since the models are mainly pre-trained on English data, we assess all the English tasks within the benchmark. These tasks include document question answering, summarization, few-shot learning, and passage retrieval. Samples in LongBench have an average input token length of 8K.

We also experiment on two additional tasks: the Needle-in-a-Haystack [28] and the GSM8k Chain-of-Thought (CoT) reasoning [56]. The Needle-in-a-Haystack test evaluates the in-context retrieval ability of long-context LLMs, asking the model to retrieve a random fact or statement placed within a lengthy document. In our experiments, we consider up to 30K document length. GSM8k is a math reasoning dataset containing 8K high quality diverse grade school math problems. Its CoT variant is a complex reasoning task that require model to attend to extensive contextual details for accurate answers, with an average input length of 3.7K.

4.1.3 ***Baselines.*** We consider H2O [63], SPARQ [46], and InfLLM [57] as our baselines. H2O is the most widely-used method of KVCache dropping and has been the basis for many enhancements. SPARQ and InfLLM are the state-of-the-art methods of KVCache offloading. In addition, we further consider a method that retrieves the exact top-$k$ tokens for each head, denoted as Oracle. In our experiments, we align the number of tokens for selective attention and the data transfer amount in Oracle, SPARQ, InfLLM, and PQCache, to achieve a fair comparison. We allow H2O to attend to more tokens, matching the memory usage of the selected key-value pairs and the data transfer amount in the other methods, following the experiment settings in SPARQ. We refer to this baseline as H2O(C), where "C" means compensation.

4.1.4 ***Hardware Environment and Hyperparameters.*** We conduct all experiments on NVIDIA A800 40GB GPU cards. Most of the hyperparameters are determined based on the number of tokens and the amount of data transferred. We use $m = 2$ and $b = 6$ for PQ by default. For other hyperparameters, we align them with the settings from the corresponding papers or open-source codes.

## 4.2 Model Performance

4.2.1 ***LongBench.*** The LongBench results of the methods on two LLMs are presented in Table 2 and 3. LongBench uses different metrics for each dataset and calculates an average score to measure overall performance. We consider including 1/5 and 1/10 of the input tokens in selective attention, respectively, with an extra communication that equals to 1/128 of the KVCache memory: for PQCache, we use $m = 2$ and $b = 6$, which satisfies $2 \times 6/16/128 < 1/128$; for SPARQ, we use $r = 1$ considering $d_h = 128$; for InfLLM, we use 1 representative token from every 128 tokens. H2O(C) is allowed to attend to more tokens as introduced in Section 4.1.3. Excluding Oracle, the best results for each setting, are highlighted in bold.

On average, models without compression (denoted as Full) can achieve the best results, since there is nearly no information loss[1]. PQCache outperforms the major baselines (i.e., H2O(C), InfLLM, and SPARQ) on most of the datasets. Although PQCache achieves slightly lower scores in a handful of cases, it exhibits substantial improvements on average. Concretely, PQCache achieves +3.88 and +6.21 improvements on Mistral-7B, and achieves +1.61 and +1.60 improvements on LLaMa2-7B, respectively. Note that the offloading-based counterparts, InfLLM and SPARQ, have the worst performance on average due to the limited additional communication for minimized latency. In contrast, PQCache performs well under the same constraint, validating the strength of our work.

Oracle is an ideal approach with the exact top-$k$ tokens for selective attention, which gives excellent performance in most cases. However, we observe that PQCache even beats Oracle and achieves the same score as the uncompressed counterpart in the "1/5#Tokens" cases. This suggests that clustering may help PQCache uncover intrinsic structures within the KVCache latent space, thus leading to promising results. Furthermore, although it is usually expected that the performance should drop when there are fewer tokens, there are exceptions where the opposite happens. This could be because not all tokens are useful for generating new ones, so getting rid of unnecessary ones might enhance inference.

4.2.2 ***Needle-in-a-Haystack.*** We use Mistral-7B-inst-v0.2 for this test. We employ the common setting [4]: the "haystack" is Paul Graham's Essays, and the "needle" is "*The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day.*" For each experiment, we use 1/5 the number of tokens in selective attention, and 1/128 extra communication. The results are shown in Figure 8, where the $x$-axis represents the "haystack" length and the $y$-axis represents the position that the "needle" hides. Greener shades indicate greater accuracy.

Among all the methods, PQCache achieves the best performance, successfully locating the needle in nearly all scenarios. The major baselines, however, fail to retrieve the needle in a substantial amount of cases. InfLLM, in particular, struggles to find the needle in most cases, possibly due to its reliance on block-partitioning and the needle not being considered as representative tokens. It can locate the needle when it is among the initial or local tokens, which we include in attention.

4.2.3 ***GSM8k CoT Reasoning.*** We use Mistral-7B-inst-v0.2 for this task. Our prompt strategy involves 8 questions with 9-step reasoning and 2 questions with 8-step reasoning per sample, a common setup for long context inference [14]. We use 1/128 extra communications. As shown in Figure 9(a), PQCache consistently outperforms H2O, SPARQ, and InfLLM under varying token counts. Some results even surpass the uncompressed counterpart, suggesting that using part of the tokens can lead to improvements. Contrary to previous findings [29], we observe that H2O performs well using the advanced Mistral model. H2O(C) performs better than PQCache using 1/10 number of tokens, as it is allowed to access more tokens.

4.2.4 ***Impact of Extra Communication.*** We investigate how the amount of extra communication impacts the model performance on the HotPotQA dataset, as shown in Figure 9(b). Fixing 1/10 tokens used, as the amount of extra communication increases from 1/128 to 1/16 of the KVCache memory, InfLLM and PQCache show relatively stable performance, while SPARQ has steadily improved

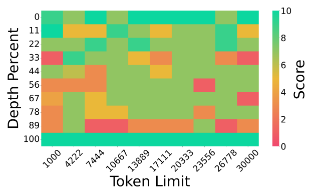
[1]In LongBench, when the sequence length exceeds the model's maximum context length, only the initial and the last tokens are used [5], resulting in information loss.

**Table 2: LongBench evaluation of the Mistral-7B-inst-v0.2 GQA model (32K context length). InfLLM, SPARQ, and PQCache all involve extra communications at an amount of 1/128 KVCache memory for pre-calculating relevance; H2O attends to more tokens where the memory equals to other methods' selected tokens and transferred data amount.**

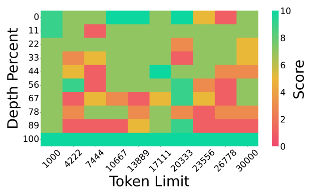
| Dataset | Full | 1/5 #Tokens + 1/128 Extra Comm | | | | | 1/10 #Tokens + 1/128 Extra Comm | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Oracle | H2O(C) | InfLLM | SPARQ | PQCache | Oracle | H2O(C) | InfLLM | SPARQ | PQCache |
| NarrativeQA | 21.27 | 22.18 | 22.07 | 19.90 | 22.25 | **22.35** | 22.34 | 21.54 | 17.55 | 22.45 | **22.62** |
| Qasper | 29.22 | 28.62 | 23.43 | 19.24 | 19.95 | **28.26** | 27.90 | 21.19 | 14.76 | 17.69 | **28.29** |
| MultiFieldQA | 47.84 | 48.02 | 43.31 | 41.13 | 39.22 | **48.27** | 48.23 | 39.22 | 36.66 | 35.02 | **47.95** |
| HotpotQA | 37.92 | 37.16 | 36.86 | 33.97 | 33.48 | **37.12** | 36.74 | 33.02 | 31.09 | 31.68 | **36.24** |
| 2WikiMQA | 21.83 | 21.02 | 18.16 | 18.48 | 16.68 | **21.25** | 21.16 | 17.76 | 16.15 | 16.14 | **21.21** |
| Musique | 18.58 | 18.45 | 17.77 | **18.96** | 16.10 | 18.37 | 18.34 | 16.85 | 14.08 | 13.18 | **18.47** |
| GovReport | 31.57 | 31.98 | 29.13 | 30.49 | 27.12 | **31.53** | 31.84 | 27.36 | 29.19 | 24.68 | **31.12** |
| QMSum | 24.31 | 23.76 | 23.23 | 22.64 | 22.21 | **23.79** | 23.98 | 23.12 | 21.52 | 22.09 | **23.27** |
| MultiNews | 26.85 | 26.79 | 25.37 | 24.38 | 23.77 | **26.70** | 26.86 | 24.94 | 23.19 | 22.40 | **26.53** |
| TREC | 71.00 | 71.00 | 68.00 | 59.50 | 62.00 | **71.00** | 71.00 | 67.50 | 53.00 | 55.50 | **70.50** |
| TriviaQA | 86.23 | 86.22 | 86.17 | 85.80 | **86.58** | 86.14 | 86.40 | **86.45** | 85.54 | 85.98 | 86.40 |
| SAMSum | 43.04 | 43.27 | 42.61 | 41.51 | 42.57 | **43.35** | 43.47 | 42.21 | 39.55 | 42.79 | **43.13** |
| Count | 2.62 | 3.42 | 3.50 | 2.96 | **4.80** | 3.93 | 3.26 | 2.44 | 2.00 | 3.15 | **3.56** |
| Retrieval | 88.74 | 88.40 | 56.56 | 35.67 | 57.19 | **88.44** | 89.80 | 37.41 | 20.00 | 39.39 | **88.63** |
| Average | 39.32 | 39.30 | 35.44 | 32.48 | 33.85 | **39.32** | 39.38 | 32.93 | 28.88 | 30.87 | **39.14** |

**Table 3: LongBench evaluation of the LLaMA-2-7B-Chat MHA model (4K context length). InfLLM, SPARQ, and PQCache all involve extra communications at an amount of 1/128 KVCache memory for pre-calculating relevance; H2O attends to more tokens where the memory equals to other methods' selected tokens and transferred data amount.**

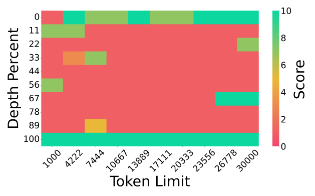
| Dataset | Full | 1/5 #Tokens + 1/128 Extra Comm | | | | | 1/10 #Tokens + 1/128 Extra Comm | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Oracle | H2O(C) | InfLLM | SPARQ | PQCache | Oracle | H2O(C) | InfLLM | SPARQ | PQCache |
| NarrativeQA | 18.78 | 18.55 | 17.53 | 12.45 | 17.40 | **19.01** | 18.05 | 17.04 | 11.88 | 16.44 | **17.56** |
| Qasper | 22.11 | 20.40 | 19.37 | 14.0 | 20.21 | **21.07** | 20.38 | 18.33 | 12.48 | 17.31 | **20.08** |
| MultiFieldQA | 36.77 | 37.58 | 31.69 | 29.39 | 33.21 | **38.72** | 37.23 | 31.82 | 23.32 | 31.27 | **37.79** |
| HotpotQA | 27.83 | 28.25 | 26.44 | 25.67 | 24.38 | **27.78** | 27.79 | 25.72 | 25.01 | 23.66 | **26.36** |
| 2WikiMQA | 31.51 | 31.96 | 29.65 | 23.95 | 29.99 | **31.23** | 31.08 | **30.68** | 25.73 | 29.02 | 30.00 |
| Musique | 8.31 | 8.23 | 8.64 | **9.03** | 7.01 | 7.65 | 8.19 | 8.40 | **8.68** | 4.91 | 7.24 |
| GovReport | 26.91 | 26.86 | 23.62 | 23.54 | 24.04 | **26.91** | 26.50 | 22.89 | 23.22 | 22.47 | **26.69** |
| QMSum | 20.68 | 20.31 | 20.97 | 19.11 | **21.03** | 20.94 | 20.57 | 20.76 | 18.69 | 20.48 | **21.33** |
| MultiNews | 26.23 | 26.08 | 24.31 | 22.32 | 25.15 | **26.46** | 26.44 | 24.28 | 20.52 | 23.48 | **26.57** |
| TREC | 64.00 | 64.00 | 62.50 | 48.0 | 62.50 | **64.00** | 64.00 | 60.50 | 40.0 | 60.50 | **63.50** |
| TriviaQA | 83.26 | 83.16 | 82.56 | 71.15 | 81.09 | **83.43** | 81.74 | 81.27 | 61.45 | 80.99 | **83.23** |
| SAMSum | 41.53 | 41.31 | 40.07 | 37.3 | 36.90 | **41.67** | 41.26 | 39.76 | 34.6 | 39.79 | **41.19** |
| Count | 2.92 | 2.98 | 2.48 | 2.68 | 2.35 | **3.03** | 3.80 | 2.68 | 2.92 | 2.58 | **3.01** |
| Retrieval | 8.00 | 7.50 | 6.50 | 6.25 | 5.50 | **7.00** | 5.50 | 4.50 | 5.00 | 6.50 | **6.50** |
| Average | 29.92 | 29.80 | 28.31 | 24.66 | 27.91 | **29.92** | 29.47 | 27.76 | 22.39 | 27.10 | **29.36** |



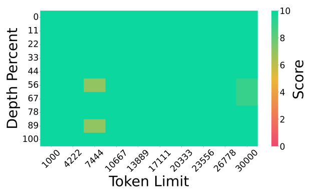
(a) H2O(C).



(b) SPARQ.



(c) InfLLM.

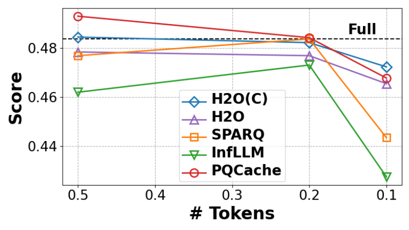


(d) PQCache.

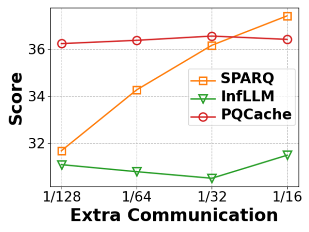
**Figure 8: Experimental results of the Needle-in-a-Haystack test.**

performance. PQCache consistently achieves high scores, outperforming the other methods when the communication amount is no larger than 1/32 KVCache. SPARQ already incurs significant latency under the 1/128 case, as shown in Section 4.3.2. Even it performs well under the 1/16 case, the latency becomes increasingly unacceptable. In low-communication scenarios, which are suitable for practical usage, PQCache achieves the best model performance.

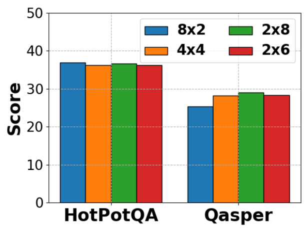
*4.2.5* ***Impact of PQ Configuration.*** We evaluate the effect of PQ configurations. Since PQ has a memory consumption of $O(s \cdot m \cdot b + 2^b \cdot d_h)$, where the second term is negligible, we adjust the values of $m$ and $b$ while keeping their product (the communication volumes) nearly unchanged. Figure 9(c) shows the results on HotPotQA and Qasper datasets with 1/10 tokens in selective attention, where the legends indicate $m \times b$. All configurations perform well.



(a) Results on GSM8k CoT.

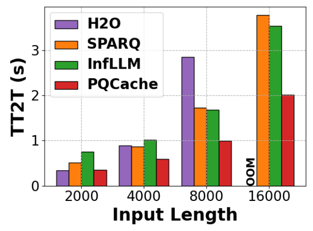


(b) Varying communications.

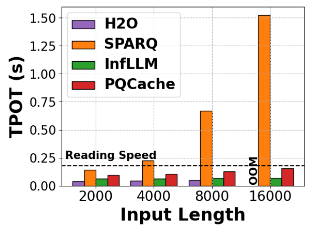


(c) Different PQ configurations.

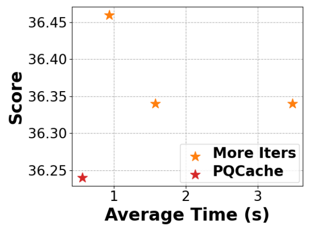
**Figure 9: Model performance on GSM8k CoT, and other hyper-parameters.**



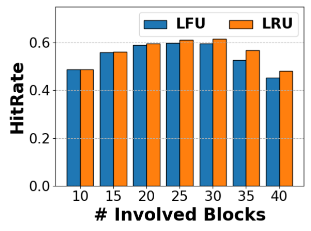
(a) Time to 2nd token.



(b) Time per output token.



(c) Trade-off time and score.



(d) Cache hit-rate.

**Figure 10: Latency experiments.**

Configurations with $m = 2$ yield the most stable results, offering an ample number of centroids. Though $b = 8$ exhibits better results, in practice we find that $b = 6$ offers more stable latency and lower memory usage while still delivering promising model performance. Therefore, we choose it as the default configuration.

## 4.3 Efficiency

*4.3.1* ***Prefilling.*** In PQCache, K-Means clustering occurs concurrently with GPU computation. While it doesn't affect the first token generation, subsequent tokens depend on clustering results. To assess system optimization, we use Time To 2nd Token (TT2T), considering query entry to LLM output time and KVCache management overhead. As shown in Figure 10(a), with overlapping and adaptive clustering, PQCache can achieve the lowest TT2T. All baseline methods have significant overhead. Since H2O collects attention scores during prefilling, it cannot utilize FlashAttention for acceleration and encounters OOM when dealing with lengthy input. SPARQ has no prefilling overhead, but its decoding process is slow (see Section 4.3.2). InfLLM incurs time overhead due to the setup required for block-level KVCache management.

*4.3.2* ***Decoding.*** Time Per Output Token (TPOT) measures the time of each decoding step. We compare the TPOT of H2O, SPARQ, InfLLM, and PQCache in Figure 10(b). Here we use 1/5 number of tokens in selective attention, and a 4096-token GPU cache. SPARQ exhibits the highest latency due to its sequential computation and communication, with the communication scaling linearly with the input sequence length. All the other methods exhibit per-token latency faster than the human reading speed, which is around 250 words (≈333 tokens) per minute [64]. H2O avoids extra communications, while InfLLM and PQCache both leverage system optimizations to accelerate decoding. InfLLM's block-level token management allows it to efficiently gather data from the CPU; however, this block-level assumption negatively impacts the model's overall quality. PQCache incorporates prefetching and caching, achieving an acceptable TPOT while not degrading model quality.

*4.3.3* ***Trade-off between Time and Accuracy.*** In Section 3.3, we design an adaptive K-Means clustering strategy to eliminate latency. To investigate the impact of this strategy on model accuracy, we conduct an experiment with varying numbers of clustering iterations on the HotpotQA dataset, with 1/10 tokens involved in attention. As shown in Figure 10(c), the adaptive strategy has the lowest clustering time with good enough model quality. Though clustering with more iterations results in better scores, the associated increase in inference latency is considerable. Trading-off time and accuracy, the adaptive strategy is the most practical choice for real-world applications. We expose an interface that lets users set the number of iterations, enabling them to balance model performance and latency for their specific needs.

*4.3.4* ***Cache Hit-rate.*** We assess the cache hit-rate for Least Recently Used (LRU) and Least Frequently Used (LFU) policies across varying numbers of top-$k_{cache}$ blocks involved during decoding. Our experiments are conducted on the HotpotQA dataset, with 1/10 tokens in selective attention and 4096 tokens in GPU cache (128 tokens per block). As shown in Figure 10(d), both LRU and LFU exhibit similar performance, achieving around 0.5 hit-rate across different numbers of blocks. As block count increases, the hit-rate initially rises due to more tokens being found within blocks. However, it eventually declines as blocks with fewer hits update the cache structure, disrupting the normal cache logic. In practice, we set the number of blocks to 32, yielding a hit-rate around 0.6, which reduces communication by 60%.

## 5 RELATED WORK

***Selective Attention for KVCache.*** To eliminate the impact of memory-intensive KVCache, a group of methods include only essential tokens for attention computation during LLM inference. One way is to discard unnecessary tokens. LM-Infinite [20] and Streaming-LLM [58] only preserve the initial tokens and the most recent tokens. H2O [63] and Scissorhands [33] utilize attention scores to identify important tokens. Their following works [1, 17, 44, 55] have explored adaptive token selection and additional metrics for better model accuracy. The LLMLingua series [26, 41] leverage an auxiliary small model to tell which tokens are necessary. Since token-level compression evicts the tokens in a greedy manner, the information loss in subsequent decoding phase may lead to model degradation. Another way is to fetch relevant tokens on demand during the decoding phase. SPARQ [46] and InfLLM [57] offload KVCache to CPU, and selectively fetch relevant key-value pairs for each attention computation. PQCache also falls under this category of methods, demonstrating effective and efficient LLM inference in comparison to existing techniques.

***KVCache Quantization.*** Quantization can be directly applied on the entire KVCache [11, 21, 34] - a straight-forward approach with promising model quality. Other compression techniques can also be employed to address the residuals introduced by quantization [29]. It is worth noting that quantization is orthogonal to token importance, and recent research has explored applying both techniques [59].

***KVCache Scheduling.*** Another way to address the KVCache memory challenge is to meticulously schedule the KVCache within memory hierarchy. FlexGen [47] employs linear programming to schedule the communication, searching for efficient patterns to store and access tensors. AttentionScore [16] maintains a hierarchical KV caching system, allowing efficient reuse of KVCache across multi-turn conversations. Another related research topic is KVCache streaming for LLM serving [32, 49], which involves handling multiple requests within more levels of memory hierarchy.

***Embedding Management.*** Embedding management is a common research focus within the database and data management domains, including embedding compression [48, 60, 62], embedding retrieval [22, 54], and key-value storage [8, 43]. Our work provides a potential direction for integrating classic embedding management into the LLM ecology.

## 6 CONCLUSION

In this paper, we proposed PQCache, a system-algorithm co-designed method for effective and efficient long context LLM inference. We incorporated the embedding retrieval technique PQ to reduce both memory and computation burden, and leveraged PQ codes and centroids to facilitate efficient MIPS for important tokens used in the attention module. Through meticulous overlapping and caching, we managed to minimize overhead to a negligible level. We evaluated PQCache on extensive experiments, and show that PQCache effectively maintains model quality with only 1/5 of the tokens involved in attention, while achieving acceptable system latency.

## REFERENCES

[1] Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: KV Cache Reduction through Key Tokens Selection for Efficient Generative Inference. CoRR abs/2403.09054 (2024). https://doi.org/10.48550/ARXIV.2403.09054 arXiv:2403.09054

[2] Moonshot AI. 2024. KimiChat. https://kimi.moonshot.cn/

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 4895–4901. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.298

[4] Anthropic. 2023. Long context prompting for Claude 2.1. https://www.anthropic.com/news/claude-2-1-prompting

[5] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. CoRR abs/2308.14508 (2023). https://doi.org/10.48550/ARXIV.2308.14508 arXiv:2308.14508

[6] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII (Lecture Notes in Computer Science), Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.), Vol. 11216. Springer, 209–224. https://doi.org/10.1007/978-3-030-01258-8_13

[7] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighborhood Search. In Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 5199–5212. https://proceedings.neurips.cc/paper/2021/hash/299dc35e747eb77177d9cea10a802da2-Abstract.html

[8] Xubin Chen, Ning Zheng, Shukun Xu, Yifan Qiao, Yang Liu, Jiangpeng Li, and Tong Zhang. 2021. KallaxDB: A Table-less Hash-based Key-Value Store on Storage Hardware with Built-in Transparent Compression. In Proceedings of the 17th International Workshop on Data Management on New Hardware, DaMoN 2021, 21 June 2021, Virtual Event, China, Danica Porobic and Spyros Blanas (Eds.). ACM, 3:1–3:10. https://doi.org/10.1145/3465998.3466004

[9] Alibaba Cloud. 2024. Tongyi Qianwen. https://tongyi.aliyun.com/qianwen/

[10] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. 2024. Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference. CoRR abs/2402.09398 (2024). https://doi.org/10.48550/ARXIV.2402.09398 arXiv:2402.09398

[11] Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. 2024. QAQ: Quality Adaptive Quantization for LLM KV Cache. CoRR abs/2403.04643 (2024). https://doi.org/10.48550/ARXIV.2403.04643 arXiv:2403.04643

[12] Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. 2019. Multi-News: A Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. In Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28-August 2, 2019, Volume 1: Long Papers, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 1074–1084. https://doi.org/10.18653/V1/P19-1102

[13] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. Proc. VLDB Endow. 12, 5 (2019), 461–474. https://doi.org/10.14778/3303753.3303754

[14] Yao Fu, Litu Ou, Mingyu Chen, Yuhao Wan, Hao Peng, and Tushar Khot. 2023. Chain-of-Thought Hub: A Continuous Effort to Measure Large Language Models' Reasoning Performance. CoRR abs/2305.17306 (2023). https://doi.org/10.48550/ARXIV.2305.17306 arXiv:2305.17306

[15] Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data Engineering for Scaling Language Models to 128K Context. CoRR abs/2402.10171 (2024). https://doi.org/10.48550/ARXIV.2402.10171 arXiv:2402.10171

[16] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. AttentionStore: Cost-effective Attention Reuse across Multi-turn Conversations in Large Language Model Serving. CoRR abs/2403.19708 (2024). https://doi.org/10.48550/ARXIV.2

403.19708 arXiv:2403.19708

[17] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. CoRR abs/2310.01801 (2023). https://doi.org/10.48550/ARXIV.2310.01801 arXiv:2310.01801

[18] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013. IEEE Computer Society, 2946–2953. https://doi.org/10.1109/CVPR.2013.379

[19] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research), Vol. 119. PMLR, 3887–3896. http://proceedings.mlr.press/v119/guo20h.html

[20] Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. LM-Infinite: Simple On-the-Fly Length Generalization for Large Language Models. CoRR abs/2308.16137 (2023). https://doi.org/10.48550/ARXIV.2308.16137 arXiv:2308.16137

[21] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. CoRR abs/2401.18079 (2024). https://doi.org/10.48550/ARXIV.2401.18079 arXiv:2401.18079

[22] Ruihong Huang, Shaoxu Song, Yunsu Lee, Jungho Park, Soo-Hyung Kim, and Sungmin Yi. 2020. Effective and Efficient Retrieval of Structured Entities. Proc. VLDB Endow. 13, 6 (2020), 826–839. https://doi.org/10.14778/3380750.3380754

[23] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. Advances in Neural Information Processing Systems 32 (2019).

[24] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. IEEE Trans. Pattern Anal. Mach. Intell. 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. CoRR abs/2310.06825 (2023). https://doi.org/10.48550/ARXIV.2310.06825 arXiv:2310.06825

[26] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 13358–13376. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.825

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. IEEE Trans. Big Data 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[28] Greg Kamradt. 2024. Needle-in-a-Haystack. https://github.com/gkamradt/LLMTest_NeedleInAHaystack

[29] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. arXiv preprint arXiv:2403.05527 (2024).

[30] Lucas D. Lingle. 2023. Transformer-VQ: Linear-Time Transformers via Vector Quantization. CoRR abs/2309.16354 (2023). https://doi.org/10.48550/ARXIV.2309.16354 arXiv:2309.16354

[31] Hao Liu, Wilson Yan, Matei Zaharia, and Pieter Abbeel. 2024. World Model on Million-Length Video And Language With Blockwise RingAttention. CoRR abs/2402.08268 (2024). https://doi.org/10.48550/ARXIV.2402.08268 arXiv:2402.08268

[32] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, et al. 2023. CacheGen: Fast Context Loading for Language Model Applications. arXiv preprint arXiv:2310.07240 (2023).

[33] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html

[34] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. CoRR abs/2402.02750 (2024). https://doi.org/10.48550/ARXIV.2402.02750 arXiv:2402.02750

[35] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[36] Julieta Martinez, Holger H. Hoos, and James J. Little. 2014. Stacked Quantizers for Compositional Vector Compression. CoRR abs/1411.2173 (2014). arXiv:1411.2173 http://arxiv.org/abs/1411.2173

[37] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards efficient generative large language model serving: A survey from algorithms to systems. arXiv preprint arXiv:2312.15234 (2023).

[38] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 1797–1807. https://doi.org/10.18653/V1/D18-1206

[39] Xiaonan Nie, Xupeng Miao, Zhi Yang, and Bin Cui. 2022. TSPLIT: Fine-grained GPU Memory Management for Efficient DNN Training via Tensor Splitting. In 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022. IEEE, 2615–2628. https://doi.org/10.1109/ICDE53745.2022.00241

[40] OpenAI. 2023. GPT-4 Technical Report. CoRR abs/2303.08774 (2023). https://doi.org/10.48550/ARXIV.2303.08774 arXiv:2303.08774

[41] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. CoRR abs/2403.12968 (2024). https://doi.org/10.48550/ARXIV.2403.12968 arXiv:2403.12968

[42] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. In 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 551–564. https://www.usenix.org/conference/atc21/presentation/ren-jie

[43] Kai Ren, Qing Zheng, Joy Arulraj, and Garth Gibson. 2017. SlimDB: A Space-Efficient Key-Value Storage Engine For Semi-Sorted Data. Proc. VLDB Endow. 10, 13 (2017), 2037–2048. https://doi.org/10.14778/3151106.3151108

[44] Siyu Ren and Kenny Q. Zhu. 2024. On the Efficacy of Eviction Policy for Key-Value Constrained Generative Language Model Inference. CoRR abs/2402.06262 (2024). https://doi.org/10.48550/ARXIV.2402.06262 arXiv:2402.06262

[45] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016. IEEE Computer Society, 18:1–18:13. https://doi.org/10.1109/MICRO.2016.7783721

[46] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2023. SparQ Attention: Bandwidth-Efficient LLM Inference. CoRR abs/2312.04985 (2023). https://doi.org/10.48550/ARXIV.2312.04985 arXiv:2312.04985

[47] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research), Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 31094–31116. https://proceedings.mlr.press/v202/sheng23a.html

[48] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 165–175. https://doi.org/10.1145/3394486.3403059

[49] Foteini Strati, Sara Mcallister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. 2024. D\'ej\aVu: KV-cache Streaming for Fast, Fault-tolerant Generative LLM Serving. arXiv preprint arXiv:2403.01876 (2024).

[50] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html 3, 6 (2023), 7.

[51] Together.ai. 2023. LLaMA-2-7B-32K. https://huggingface.co/togethercomputer/LLaMA-2-7B-32K

[52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan 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 Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem 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, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. CoRR abs/2307.09288 (2023). https://doi.org/10.48550/ARXIV.2307.09288 arXiv:2307.09288

[53] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural Discrete Representation Learning. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 6306–6315. https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html

[54] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. Proc. VLDB Endow. 14, 11 (2021), 1964–1978. https://doi.org/10.14778/3476249.3476255

[55] Zihao Wang and Shaoduo Gan. 2024. SqueezeAttention: 2D Management of KV-Cache in LLM Inference via Layer-wise Optimal Budget. arXiv preprint arXiv:2404.04793 (2024).

[56] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[57] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, Song Han, and Maosong Sun. 2024. InfLLM: Unveiling the Intrinsic Capacity of LLMs for Understanding Extremely Long Sequences with Training-Free Memory. CoRR abs/2402.04617 (2024). https://doi.org/10.48550/ARXIV.2402.04617 arXiv:2402.04617

[58] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient Streaming Language Models with Attention Sinks. CoRR abs/2309.17453 (2023). https://doi.org/10.48550/ARXIV.2309.17453 arXiv:2309.17453

[59] June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024. No Token Left Behind: Reliable KV Cache Compression via Importance-Aware Mixed Precision Quantization. CoRR abs/2402.18096 (2024). https://doi.org/10.48550/ARXIV.2402.18096 arXiv:2402.18096

[60] Hailin Zhang, Zirui Liu, Boxuan Chen, Yikai Zhao, Tong Zhao, Tong Yang, and Bin Cui. 2024. CAFE: Towards Compact, Adaptive, and Fast Embedding for Large-scale Recommendation Models. Proceedings of the ACM on Management of Data 2, 1 (2024), 1–28.

[61] Hailin Zhang, Yujing Wang, Qi Chen, Ruiheng Chang, Ting Zhang, Ziming Miao, Yingyan Hou, Yang Ding, Xupeng Miao, Haonan Wang, Bochen Pang, Yuefeng Zhan, Hao Sun, Weiwei Deng, Qi Zhang, Fan Yang, Xing Xie, Mao Yang, and Bin Cui. 2023. Model-enhanced Vector Index. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/ac112e8ffc4e5b9ece32070440a8ca43-Abstract-Conference.html

[62] Hailin Zhang, Penghao Zhao, Xupeng Miao, Yingxia Shao, Zirui Liu, Tong Yang, and Bin Cui. 2023. Experimental Analysis of Large-scale Learnable Vector Storage Compression. Proc. VLDB Endow. 17, 4 (2023), 808–822. https://www.vldb.org/pvldb/vol17/p808-zhang.pdf

[63] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html

[64] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. CoRR abs/2401.09670 (2024). https://doi.org/10.48550/ARXIV.2401.09670 arXiv:2401.09670