FreeKV: Boosting KV Cache Retrieval for Efficient LLM Inference

Guangda Liu[1] **Chengwei Li**[1] **Zhenyu Ning**[1] **Jing Lin**[2] **Yiwu Yao**[2] **Danning Ke**[2]
Minyi Guo[1] **Jieru Zhao**[1*]
[1] School of Computer Science, Shanghai Jiao Tong University [2] Huawei Technologies Co., Ltd

Abstract

Large language models (LLMs) have been widely deployed with rapidly expanding context windows to support increasingly demanding applications. However, long contexts pose significant deployment challenges, primarily due to the KV cache whose size grows proportionally with context length. While KV cache compression methods are proposed to address this issue, KV dropping methods incur considerable accuracy loss, and KV retrieval methods suffer from significant efficiency bottlenecks. We propose **FreeKV**, an algorithm-system co-optimization framework to enhance KV retrieval efficiency while preserving accuracy. On the algorithm side, FreeKV introduces speculative retrieval to shift the KV selection and recall processes out of the critical path, combined with fine-grained correction to ensure accuracy. On the system side, FreeKV employs hybrid KV layouts across CPU and GPU memory to eliminate fragmented data transfers, and leverages double-buffered streamed recall to further improve efficiency. Experiments demonstrate that FreeKV achieves near-lossless accuracy across various scenarios and models, delivering up to 13$\times$ speedup compared to SOTA KV retrieval methods.

1 Introduction

Large language models (LLMs) have gained remarkable prominence for their ability to excel across diverse tasks an have been widely deployed in a variety of applications, such as document analysis, chatbot and coding assistant [1, 2, 3]. To process increasingly complex tasks such as long-document QA, multi-turn dialogue and repository-level code understanding, the context window sizes of LLMs are rapidly expanding to accommodate longer inputs. Mainstream LLMs now support context windows of 128K tokens [4, 5], with frontier models reaching up to 1 million tokens [6, 7].

While larger context windows unlock new capabilities for applications, handling long context presents significant challenges for efficient deployment. These challenges arise from the KV cache in LLMs, which stores the key-value states of previous tokens to avoid recomputation during inference, causing its size to grow proportionally with the context length. On the one hand, the size of KV cache can exceed the capacity of GPU memory. For instance, the KV cache for a single request can reach 40GB for Llama-3-70B with a context length of 128K [8]. On the other hand, since the LLM decoding is memory-bound, accessing a large KV cache significantly degrades the decoding speed [9].

To mitigate these issues, based on the sparsity of attention computation, previous works proposed compressing the KV cache, i.e., utilizing only a portion of the KV cache for inference. The compression methods can be broadly classified into two categories: **KV dropping** and **KV retrieval** [10]. KV dropping methods only retain KV cache for important tokens and permanently evict unimportant ones. The identification of important tokens can be performed either statically [11, 12, 13] or dynamically [14, 15, 16]. In contrast, KV retrieval methods maintain the entire KV cache but dynamically select a subset for inference [17, 18, 19, 20, 21].

KV dropping：丢弃一部分KV对省下内存
KV retrieval:生成时只使用最相关的一部分KV

*Correspondence to Jieru Zhao (zhao-jieru@sjtu.edu.cn)

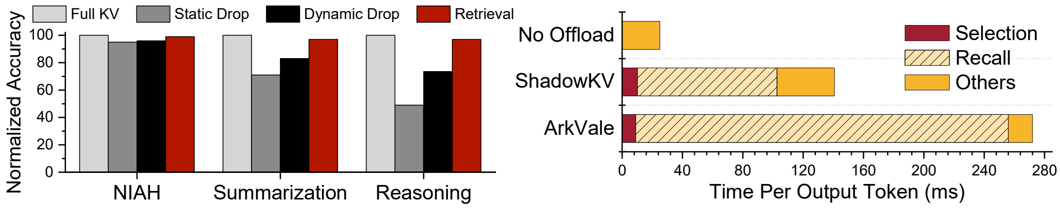
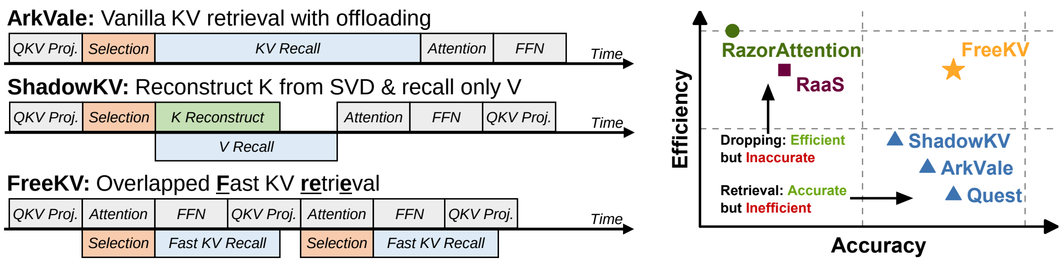


Figure 1: Left: Accuracy comparison of KV dropping and retrieval methods across different tasks. Right: Latency breakdown of KV retrieval methods with offloading.



(a) (b)

Figure 2: (a) Comparison of timelines for KV retrieval methods, FreeKV shifts the selection and recall out of the critical path. (b) Accuracy-efficiency trade-off of KV compression methods.

While both KV dropping and retrieval methods can maintain acceptable model accuracy under specific scenarios and tasks, recent studies reveal significant accuracy degradation with KV dropping methods, particularly on tasks like summarization and reasoning [22, 23]. This degradation stems from the dynamic nature of token importance, where tokens previously deemed unimportant and permanently dropped may become crucial in later steps [18, 21]. For complex tasks involving long generation, the omission of a large number of such important tokens results in severe accuracy decline. This issue is further exacerbated with the advent of reasoning models, where extended thinking processes lead to generation lengths reaching 32K tokens or more [24, 5, 25], highlighting the limitations of KV dropping methods. In Fig. 1, we compare the accuracy of static KV dropping (RazorAttention [12]), dynamic KV dropping (RaaS [16]) and KV retrieval (Quest [17]) under similar KV cache budgets, across tasks of Needle In A Haystack (NIAH), summarization and reasoning [26, 27, 28]. Both static and dynamic drop methods exhibit significant accuracy degradation on summarization and reasoning tasks. In contrast, KV retrieval methods maintain robust accuracy across all tasks. Therefore, **KV retrieval methods are better suited for more general and practical scenarios.**

Despite their superior accuracy performance, **KV retrieval methods face significant efficiency challenges**. First, since the complete KV cache must be retained, retrieval methods often offload the KV cache to CPU memory to circumvent GPU memory limitations. For methods without offloading like Quest [17], out-of-memory error is inevitable for long contexts and large batch sizes. However, for offloading methods, due to the low bandwidth of the CPU-GPU connection, *recalling* the selected KV tuples from CPU memory to GPU memory incurs long latency. Second, KV retrieval methods select KV tuples from the entire context, leading to considerable *selection* overhead, even though most retrieval methods adopt page-wise selection to alleviate this issue. In Fig. 1, we present the latency breakdown of SoTA offloading KV retrieval methods, using Llama-3.1-8B-Instruct with a batch size of 1 and a context length of 32K. For ArkVale [18], recall and selection contribute approximately 94% of the overall latency. Similarly, while ShadowKV [19] introduces key cache reconstruction to recall only the value cache, recall and selection still comprise about 73% of the total latency. Both methods lead to significantly higher latency compared to inference with the full KV cache without offloading.

To overcome these challenges, we introduce FreeKV, an algorithm-system co-optimization framework that significantly boosts the efficiency of KV retrieval, while maintaining near-lossless model accuracy across diverse scenarios and tasks. **On the algorithm side**, leveraging the high similarity of query vectors between adjacent decoding steps, FreeKV introduces *speculative retrieval*, which shifts the selection and recall processes out of the critical path via step-wise KV reuse, thus avoiding inference blocking. As illustrated in Fig. 2a, this approach allows selection and recall to overlap with other

Table 1: Comparison of KV cache compression methods.

	RazorAttn	RaaS	Quest	ArkVale	ShadowKV	FreeKV
Category	Static Drop	Dynamic Drop	Retrieval	Retrieval	Retrieval	Retrieval
Long Generation	✔	✔	✔	✔	✘	✔
GPU Mem. Usage	$O(\mathbf{s}L)$	$O(\mathcal{B})$	$O(L)$	$O(\mathcal{B})$	$O(\frac{r}{d_{kv}}L+\mathcal{B})$	$O(\mathcal{B})$
Group-consistent	✔	✘	✘	✔	✔	✔
Efficiency	High	High	Low	Low	Low	High

operations, effectively hiding their overhead. To counter potential accuracy losses from pure KV reuse, FreeKV incorporates *fine-grained correction* to preserve model accuracy with minimal impact on efficiency. **On the system side**, FreeKV employs *hybrid KV layouts* across CPU and GPU memory to eliminate inefficient fragmented data transfers and avoid layout conversion overhead during inference. In addition, FreeKV implements a double-buffering mechanism to facilitate *streamed recall*, further improving recall efficiency by overlapping CPU-GPU and GPU-GPU data transfers. As shown in Fig. 2b, FreeKV strikes a balance between accuracy and efficiency, establishing a new Pareto frontier. Extensive experiments show that FreeKV maintains near-lossless accuracy across diverse scenarios and models, delivering up to 13× speedup over SOTA KV retrieval methods.

2 Background and related work

2.1 Problem formulation

The decoding process of an attention head in LLMs can be expressed as $\mathbf{o} = \text{softmax}(\mathbf{q}\mathbf{K}^T/\sqrt{d})\mathbf{V}$, where $\mathbf{q}, \mathbf{o} \in \mathbb{R}^{1\times d}$ are the query vector and attention output of the current token, respectively, and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times d}$ represent the K and V states of the L preceding tokens. For modern LLMs with Grouped Query Attention (GQA) [29], the number of attention heads and KV heads are denoted as n_{qo} and n_{kv}, respectively. The output of attention head h is given by $\mathbf{o}_h = \text{softmax}(\mathbf{q}_h\mathbf{K}_\hbar^T/\sqrt{d})\mathbf{V}_\hbar$, where $\hbar = \frac{h}{n_{qo}/n_{kv}} = \frac{h}{G}$ is the corresponding KV head. And $G = \frac{n_{qo}}{n_{kv}}$ is the group size, representing the number of attention heads within a group that share the same KV head.

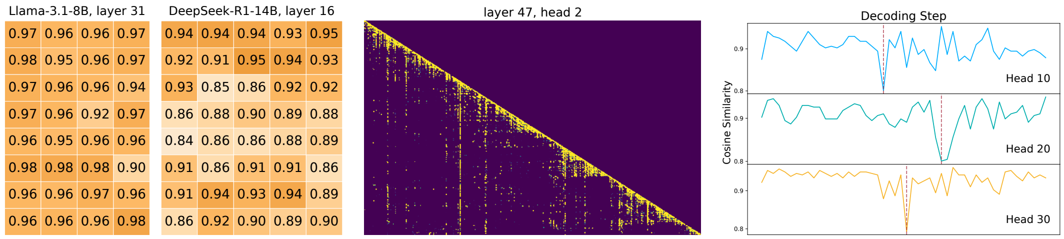
KV retrieval methods select a subset of KV tuples for attention computation based on attention weights derived from $\mathbf{q}$ and $\mathbf{K}$, defined as $\mathcal{I}^h = \mathit{Sel}(\mathbf{q}^h, \mathbf{K}^\hbar)$, where $\mathcal{I}^h$ represents the indices of selected KV tuples, and $|\mathcal{I}^h| = \mathcal{B}$ specifies a preset KV cache budget. In practice, retrieval methods consistently retain KV tuples for $\mathcal{S}$ sink tokens at the beginning and $\mathcal{W}$ tokens within the local window, leaving $\mathcal{B} - \mathcal{S} - \mathcal{W}$ tuples available for selection.

For GQA models, the space required for retrieved KV tuples is $O(\mathcal{B} \times n_{kv})$ if the selection is *group-consistent*, meaning the indices of KV tuples selected by all attention heads within the same group are identical, i.e., $\mathcal{I}^{(m-1)\times G+1} = \mathcal{I}^{(m-1)\times G+2} = \cdots = \mathcal{I}^{m\times G}$ for $m = 1, 2, \ldots, n_{kv}$. However, if the selection is not group-consistent, the required space increases to $O(\mathcal{B} \times n_{qo})$, resulting in G times higher costs in both space and memory accesses.

2.2 Related work

KV dropping KV dropping methods can be further categorized into static and dynamic dropping. Static dropping methods evict KV states using fixed patterns determined before inference. For instance, StreamingLLM [11] retains KV tuples only for the initial *sink* tokens and those within a local window. Based on this, RazorAttention [12] and DuoAttention [13] retain full KV cache for designated *retrieval heads*, while limiting other heads to sink tokens and a local window. Static dropping methods are computationally efficient, incurring minimal overhead during inference, with GPU memory usage proportional to a preset sparsity $\mathbf{s}$ and the context length L. However, their fixed nature overlooks dynamic patterns during inference, leading to significant accuracy losses. Dynamic dropping methods, on the other hand, evict KV tuples based on attention scores calculated online during inference [14, 15]. While most dropping methods do not support the long-generation scenarios, RaaS [16] addresses these scenarios by evicting tokens that have not received significant attention scores for a sustained period. Dynamic dropping methods retain and score only a fixed budget of the KV cache, achieving good efficiency despite the additional scoring overhead.

KV retrieval Both static and dynamic dropping methods incur permanent information losses, resulting in notable accuracy degradation, particularly in long-generation scenarios. In contrast,



(a) (b) (c)

Figure 3: (a) Cosine similarities between query vectors of adjacent generated tokens, averaged over generation; each cell corresponds to an attention head. (b) Attention map of DeepSeek-R1-Qwen-14B on reasoning tasks. (c) Variations in $\mathcal{C}_i$ during generation of DeepSeek-R1-Qwen-14B.

KV retrieval methods retain the complete KV cache but select a subset for computation. While preserving accuracy, retrieval methods introduce significant efficiency challenges. First, **applying selection across the entire context by scoring over every token leads to unacceptable overhead**. To mitigate this, most KV retrieval methods adopt **page-wise selection**, summarizing the keys within a page and scoring only these *page summaries*. For example, Quest [17] uses min-max pooled keys, ArkVale [18] employs bounding volumes of keys within a page, and ShadowKV[19] simply relies on mean-pooled keys. Moreover, **the handling of the complete KV cache poses significant challenges**. Quest stores the entire KV cache in GPU memory with limited capacity, restricting support for long context lengths and large batch sizes. Furthermore, its inconsistent selection within head groups incurs G times memory access overhead. ArkVale offloads the KV cache to CPU memory and recalls the selected KV pages during inference. While ensuring group-consistent using mean pooling over attention weights and maintaining a cache for selected pages on GPU, the recall process of ArkVale remains costly, severely impacting efficiency. ShadowKV takes a different approach by leveraging the low-rank property of the pre-rope key cache. It retains only the low-rank key cache obtained through singular value decomposition (SVD). During inference, for selected pages, it reconstruct the key cache from the low-rank representations, while only recalling the value cache. This reduces memory transfer costs but requires additional GPU memory to store the low-rank key, consuming $\frac{r}{d_{kv}}$ (15%-30%) of the original key cache size, where $r = 160$ is the rank used by ShadowKV and d_{kv} is the dimension of key cache. Moreover, ShadowKV does not support long-generation since the SVD is performed only once during prefill, leaving the low-rank key unupdated during decoding.

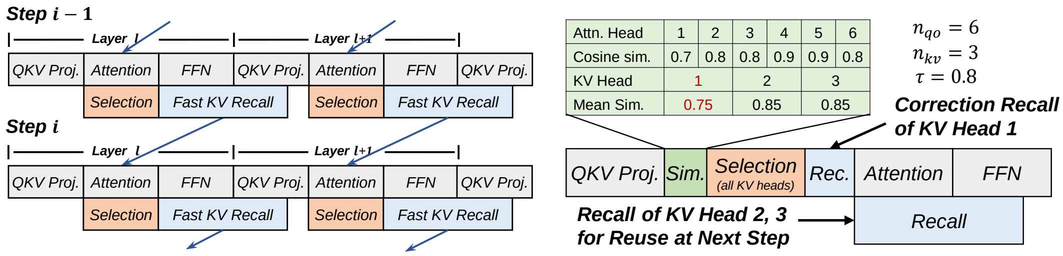
The features of KV dropping and retrieval methods are summarized in Table 1. As illustrated, FreeKV ensures accuracy preservation through KV retrieval while attaining high efficiency with fixed $O(\mathcal{B})$ GPU memory usage and group-consistent selection.

3 Algorithm design

3.1 Observation

We sample the query vectors of generated tokens during inference of Llama-3.1-8B-Instruct on long-generation tasks and DeepSeek-R1-Qwen-14B on long-reasoning tasks. The cosine similarity between the query vectors of adjacent generated tokens is calculated as $\mathcal{C}_i = \frac{\langle \mathbf{q}_i, \mathbf{q}_{i-1} \rangle}{|\mathbf{q}_i| \cdot |\mathbf{q}_{i-1}|}$, where $\mathbf{q}_{i-1}, \mathbf{q}_i \in \mathbb{R}^{1\times d}$ are the query vectors of tokens generated at step $i-1$ and i. We present the mean similarity during generating $g = 13000$ tokens, calculated as $\sum_{i=1}^{g} \frac{\mathcal{C}_i}{g}$, in Fig. 3a. As shown, across various models and tasks, the mean similarity of all attention heads consistently exceeds 0.84, with most heads achieving a similarity greater than 0.9. This high similarity is observed across different layers, models, and tasks, likely due to position embeddings [30] and the semantic continuity of adjacent tokens [31]. This observation aligns with the vertical line patterns in attention maps, as shown in Fig. 3b and supported by prior studies [32, 16, 33], which show that adjacent decoding steps exhibit high attention scores on similar tokens. This insight motivates the **speculative recall** mechanism (Sec. 3.2) of FreeKV. 观察1：不同模型不同任务注意力头相似度都高

To delve deeper, we analyze the changes in similarity during generation for DeepSeek-R1-Qwen-14B on reasoning tasks. As shown in Fig. 3c, while the mean similarity remains high, certain decoding steps exhibit outliers with significantly lower similarity. Moreover, these outlier steps vary across



(a) (b)

Figure 4: (a) Inference timeline with speculative retrieval, where the blue arrows represent the reuse of KV pages recalled in the previous step. (b) Timeline for fine-grained correction with query-based identification and head-wise correction recall.

attention heads, indicating head-specific variations in query similarity during decoding. These variations underpins the **fine-grained correction** mechanism (Sec. 3.3) employed by FreeKV.

观察2：Q的相似度会随特定的头改变

3.2 Speculative retrieval

相邻step中q和token相似

Speculative retrieval Based on the high similarity observed in query vectors and selected tokens between adjacent decoding steps, i.e., $Sel(\mathbf{q}_i, \mathbf{K}) \sim Sel(\mathbf{q}_{i-1}, \mathbf{K})$, we propose a speculative retrieval mechanism that shifts the selection and recall out of the critical path of inference. Specifically, the attention computation of step i bypasses the selection and recall, instead directly being launched by reusing the KV tuples recalled during step $i-1$, as shown in Fig. 4a. This design enables the selection and recall operations to overlap with attention and FFN computations of the current layer, as well as the QKV projections of the next layer. The recalled KV tuples during step i will then be reused in step $i+1$, continuing the process iteratively. 直接复用上一步的KV 代替召回之前的

Group-consistent selection FreeKV adopts page-wise selection, utilizing the min-max pooled keys within each page as the page summary, similar to Quest [17]. Let n_{page} denote the number of KV pages. To ensure group-consistent selection, after computing the attention weights $\mathcal{P}^h \in \mathbb{R}^{n_{\text{page}}}$ for query vector of attention head h and the corresponding page summaries, FreeKV applies mean pooling across the group over softmax$(\mathcal{P}^h)$. For KV head m, the corresponding attention heads select consistent pages based on scores calculated as $\sum_{j=1}^{G} \text{softmax}(\mathcal{P}^{(m-1)\times G+j})/G$ 这个式子计算每个page的注意力分数

KV以page为单位存储，然后对于每组attention head，要选取一致的KV页（平均分数最高的）

3.3 Fine-grained correction

弥补准确度损失的方法

While purely reusing KV pages recalled from the previous step maximizes efficiency, it can result in significant accuracy degradation. To mitigate this, FreeKV introduces a correction mechanism that selectively recall KV pages for the current step. By employing query-based identification and head-wise recall, FreeKV minimizes the associated efficiency overhead.

Query-based identification A straightforward correction involves directly comparing the indices of selected KV tuples between step i and $i-1$, i.e., $Sel(\mathbf{q}_i, \mathbf{K})$ and $Sel(\mathbf{q}_{i-1}, \mathbf{K})$. However, this approach incurs substantial overhead due to index comparisons and hinders the overlap of selection with other operations. To address these limitations, FreeKV employs a correction mechanism based on the cosine similarity of query vectors, $\mathcal{C}_i$. Correction is triggered only if $\mathcal{C}_i < \tau$, indicating a significant deviation of $Sel(\mathbf{q}_i, \mathbf{K})$ from $Sel(\mathbf{q}_{i-1}, \mathbf{K})$, where τ is a predefined threshold. To ensure group consistency, FreeKV performs mean pooling over $\mathcal{C}_i$ across the group, and compares the pooled value with τ to determine whether correction is required for a KV head. As illustrated in Fig. 4b, KV head 1, with a mean similarity of 0.75 (below $\tau = 0.8$), is flagged for correction.

第i步和第i-1步相似度过低时触发校正，不过是以组为单位判断这一组所用的KV是否需要校正

Head-wise correction As shown in Fig 4b, once the KV heads requiring correction are identified, FreeKV initiates selection and recall for these KV heads before the attention computation at the current decoding step. For KV heads that do not require correction, recall is deferred and overlapped with other operations, retrieving selected KV tuples for the reuse at the next decoding step. To avoid the overhead and reduce GPU utilization caused by separately launching selection for corrected and non-corrected heads, FreeKV executes selection for all KV heads whenever correction is required. Then for non-corrected KV heads, the recall proceeds directly without repeating the selection.

小技巧：在确定需要校正的头之后不用重新做选择操作，而是先校正再一起recall

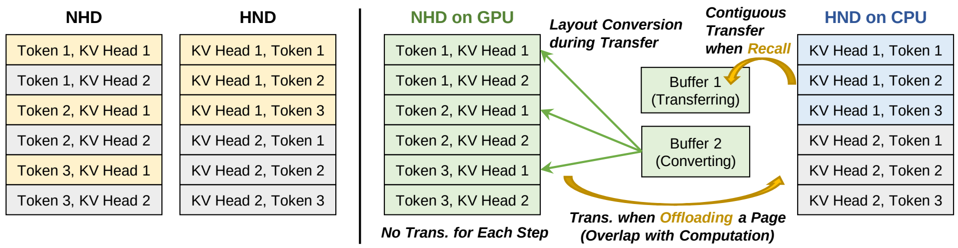


Figure 6: Left: KV cache pages under NHD and HND layouts with $p = 3$ and $n_{kv} = 2$; the highlights represent elements for a given KV head. Right: Hybrid KV cache layouts across CPU and GPU memory, along with streamed recall enabled by double-buffering.

4 System design and implementation 主要是实现上的一些细节 内存管理性能优化之类的

Effective recall overlapping to minimize overhead demands high recall efficiency. FreeKV achieves this through a dedicated system design and implementation, featuring caching, hybrid layouts and streamed recall, as detailed in the following sections.

4.1 Overview

The system overview of FreeKV is illustrated in Fig. 5. In the data plane, FreeKV retains the query vectors from the previous step, page summaries and cache for selected KV pages in GPU memory. In CPU memory, FreeKV maintains a complete KV cache pool for offloading KV pages. In the control plane, a controller on CPU manages the scheduling and synchronization of operations such as correction, attention, selection and recall, following the timeline described in Section 3.

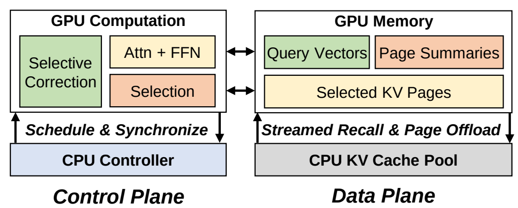


Figure 5: System overview of FreeKV

4.2 Hybrid layouts and streamed recall

The KV cache layout defines the memory organization of the underlying key-value tensors. Two commonly used KV cache layouts are NHD and HND [34]. The NHD layout organizes the KV cache in the shape of (L, n_{kv}, d), while the HND layout uses the shape of (n_{kv}, L, d). In practice, when managing the KV cache in pages, the shapes of NHD and HND layouts are $(n_{\text{page}}, p, n_{kv}, d)$ and $(n_{\text{page}}, n_{kv}, p, d)$, respectively, where p is the page size. Since the key and value derived from projections over hidden states are $K, V \in \mathbb{R}^{L\times(n_{kv}\times d)}$, NHD is the natural layout while the HND layout requires additional transpose operations. To eliminate this overhead, mainstream efficient inference frameworks adopts the NHD layout [35].

However, since the indices of selected KV pages differ across KV heads and recall is performed individually for each KV head, using the NHD layout results in inefficient fragmented data transfers. As shown on the left side of Fig. 6, under the NHD layout, for a given KV head, the memory of $p = 3$ key/value vectors within a page is non-contiguous. When recalling a key/value page, the maximum transfer unit contains only d elements, equivalent to just 256 bytes for $d = 128$ and Float16 precision. This extensive fragmented data transfers significantly degrade recall efficiency. In contrast, the HND layout ensures that p key/value vectors within a page are contiguous for each KV head, allowing a transfer unit of $p \times d$ elements, or 8KB when $p = 32$.

Hybrid layouts To avoid fragmented data transfer while minimizing transpose overhead, FreeKV adopts hybrid layouts on CPU and GPU memory. As illustrated in Fig. 6, FreeKV employs the NHD layout on GPU to eliminate the need for per-step transposes during decoding, and the HND layout on CPU to ensure contiguous and efficient CPU-GPU data transfers during recall. With the hybrid layouts, the NHD-HND transpose is only required when offloading a KV page, effectively amortizing the overhead. In addition, FreeKV utilize an HND layout on CPU with a shape of $(n_{\text{page}}, n_{kv}, 2, p, d)$, enabling the transfer of $2 \times p \times d$ contiguous elements for both key and value vectors during recall.
CPUGPU混合布局

Streamed recall While offloading and the associated transposes can overlap with computation, the conversion from HND layout to NHD layout during recall can block data transfers and subsequent attention computation. To avoid such blocking from sequential data transfer and layout conversion,

Table 2: Accuracy results of LongBench v2 and LongGenBench.

Methods	LongBench v2				LongGenBench		
	Overall	Short	Medium	Long	CR	Acc	CR×Acc
Llama-3.1-8B-Instruct	29.22	34.44	27.91	23.15	80.03	33.52	26.82
RazorAttention	27.44	33.89	25.12	21.30	35.90	34.01	12.20
RaaS	28.23	33.89	26.51	22.02	76.63	33.93	26.00
Quest	28.43	33.33	**27.44**	22.22	78.03	35.40	**27.71**
ArkVale	**28.63**	**33.89**	26.98	**23.15**	39.36	26.33	10.36
ShadowKV	25.45	32.78	22.79	18.52	**79.28**	38.68	**30.66**
FreeKV	**29.22**	**35.00**	**27.44**	**23.15**	**78.03**	35.40	27.62
Qwen-2.5-7B-Instruct	27.44	36.11	23.72	20.37	79.56	39.08	31.09
RazorAttention	25.25	32.78	21.86	19.44	42.13	50.99	21.48
RaaS	26.24	35.56	21.86	19.44	**77.65**	44.31	**34.40**
Quest	**27.63**	**36.67**	**22.79**	22.22	62.89	41.28	25.96
ArkVale	26.84	**36.11**	22.33	20.37	75.91	41.89	31.79
ShadowKV	25.84	32.22	20.00	**26.85**	35.49	32.22	11.43
FreeKV	**26.84**	34.44	**22.33**	**23.15**	**76.93**	42.66	**32.81**
Qwen-2.5-14B-Instruct	33.40	41.11	31.16	25.00	65.84	44.58	29.35
RazorAttention	34.19	**43.33**	30.70	**25.93**	26.48	52.46	13.89
RaaS	32.60	40.56	32.09	20.37	**62.29**	47.83	**29.79**
Quest	33.80	40.00	33.49	24.07	45.49	43.45	19.76
ArkVale	34.19	41.11	33.49	24.07	45.31	43.37	19.65
ShadowKV	**34.79**	40.56	**34.88**	**25.00**	21.25	38.85	8.25
FreeKV	**34.19**	**41.11**	**33.49**	24.07	**65.46**	44.90	**29.39**

FreeKV employs a *double-buffering* mechanism to achieve streamed recall. As shown in Fig. 6, after a selected KV page is transferred to buffer 2, its layout conversion begins immediately, while the transfer of the next page is concurrently initiated into buffer 1. Both buffers and the conversion process reside in GPU memory, leveraging its high bandwidth to enhance efficiency.

双缓冲区一起实现召回

5 Evaluation

5.1 Experimental setup

Datasets and models We evaluate FreeKV across various models and tasks. For accuracy evaluation, we select LongBench v2 [36] and LongGenBench [37] to cover long-input and long-generation scenarios. In addition, we assess FreeKV on long reasoning tasks, including MATH500 [38], AIME24 [28] and GPQA [39]. For LongBench v2 and LongGenBench, we use general models including Llama-3.1-8B-Instruct [8], Qwen-2.5-7B-Instruct and Qwen-2.5-14B-Instruct [4]. For reasoning tasks, we use DeepSeek-R1-Llama-8B, DeepSeek-R1-Qwen-7B and DeepSeek-R1-Qwen-14B [5]. Detailed metrics of each dataset are provided in the corresponding sections.

Baselines We compare FreeKV against SOTA methods, including static KV dropping such as RazorAttention [12], dynamic KV dropping methods like RaaS [16], and KV retrieval methods, including Quest [17], ArkVale [18] and ShadowKV [19]. The sparsity of RazorAttention is set to 0.15, while the budget $\mathcal{B}$ for all other methods is consistently set to 2048. The sink size $\mathcal{S}$, local window size $\mathcal{W}$ and the correction threshold τ of FreeKV vary depending on the task. Since the original implementations of RaaS and Quest are not group-consistent, we adapt them by applying maximum pooling over scores within the group to ensure consistent selection. For ShadowKV, which does not natively support long-generation scenarios, we modify it to update the SVD results every $\mathcal{W}$ generated tokens. Following standard practice, KV cache compression is not applied to the first layer in any of the methods. And we consistently set the page size to 32 for FreeKV, Quest, ArkVale, ShadowKV and RaaS. Other hyperparameters of the baselines, such as the update threshold for RaaS and the SVD rank for ShadowKV, are retained as specified in their original configurations.

5.2 Accuracy evaluation

LongBench v2 Improved from LongBench [27], LongBench v2 covers more realistic scenarios. It spans various difficulty levels and context lengths, ranging from 8K to 2M tokens. All problems of LongBench v2 are presented in a multi-choices question format, with accuracy used as the unified metric. We report accuracy under the context length categories of *short*, *medium* and *long*, as well as the overall accuracy. For all methods, we truncated the inputs to 64K tokens, set $\mathcal{S} = \mathcal{W} = 128$ and $\tau = 0.8$ and applied greedy decoding

As shown in the left part of Table 2, for all models, the overall accuracy of FreeKV deviates by at most 0.6 compared to the model with full KV cache, while FreeKV achieves the best or second-best

Table 3: Accuracy results of long reasoning tasks.

Methods	MATH500		AIME24		GPQA	
	pass@k	**avg@k**	**pass@k**	**avg@k**	**pass@k**	**avg@k**
DeepSeek-R1-Llama-8B	78.00	67.25	80.00	47.08	82.00	39.75
RazorAttention	72.00	60.50	46.67	30.00	60.00	34.25
RaaS	74.00	62.50	66.67	36.25	64.00	33.50
Quest	72.00	62.00	73.33	44.17	76.00	37.25
ArkVale	72.00	**62.50**	**80.00**	**46.67**	72.00	**39.75**
ShadowKV	**76.00**	60.25	63.33	36.50	**78.00**	36.25
FreeKV	**78.00**	**66.75**	**76.67**	**47.50**	**86.00**	**41.25**
DeepSeek-R1-Qwen-7B	78.00	71.75	83.33	56.66	72.00	35.75
RazorAttention	72.00	66.75	65.33	35.42	60.00	32.50
RaaS	74.00	67.00	73.33	42.92	58.00	33.25
Quest	76.00	68.00	**76.67**	47.50	**72.00**	**38.75**
ArkVale	**76.00**	**68.25**	73.33	**47.92**	72.00	34.25
ShadowKV	74.00	64.75	73.33	43.75	70.00	33.50
FreeKV	**78.00**	**70.00**	**83.33**	**52.92**	**74.00**	**39.50**
DeepSeek-R1-Qwen-14B	74.00	70.25	86.67	66.25	82.00	53.25
RazorAttention	70.00	59.75	46.67	32.50	68.00	38.50
RaaS	68.00	64.75	73.33	48.75	80.00	44.25
Quest	**76.00**	**67.25**	**83.33**	58.33	80.00	51.25
ArkVale	72.00	66.25	76.67	**61.25**	**86.00**	**53.75**
ShadowKV	76.00	65.00	83.33	57.25	86.00	51.75
FreeKV	**78.00**	**67.50**	**83.33**	**64.17**	**86.00**	**56.00**

performance across most metrics. KV dropping methods, although exhibiting moderate accuracy losses on this long-input benchmark, consistently underperform compared to KV retrieval methods.

LongGenBench Unlike traditional long-context benchmarks that focus on long inputs, LongGenBench is designed to evaluate the model's ability to handle long generations, assessing the its capability to generate coherent and high-quality long-form content. Each task of LongGenBench contains subtasks that prompt the model to generate specific content at designated points, within specific ranges or in a periodic manner. We report the completion rate (**CR**) of subtasks, the accuracy of completed subtasks (**Acc**), and the overall accuracy (**CR** $\times$ **Acc**). As LongGenBench relies on LLMs for accuracy evaluation, we use Qwen-3-32B [40] as the evaluator. For all experiments, we set $\mathcal{S} = \mathcal{W} = 512$ and $\tau = 0.9$, applying stochastic sampling with a temperature of 0.95, a top-p value of 0.95, and a maximum generation length of 16K following the original setup.

As shown in the right part of Table 2, across all evaluated models, FreeKV maintains overall accuracy comparable to or exceeding that of the model with full KV cache. Compared to other methods, FreeKV achieves the best or second-best performance in terms of CR and overall accuracy. For long-generation tasks, RazorAttention with static dropping suffers significant accuracy losses, while RaaS with dynamic dropping demonstrates strong accuracy, likely due to the relative simplicity of the tasks. In addition, we observe repeated output and reduced accuracy for ShadowKV with Qwen-2.5 models, which can be attributed to errors in the reconstructed keys.

Long reasoning tasks In addition to the general long-generation tasks where users prompt to generate long-form content, reasoning models like DeepSeek-R1 autonomously generate long thinking processes to solve complicated problems. We evaluate reasoning tasks using problems from MATH500, AIME24 and GPQA datasets, which covers a range of difficulties in mathematical reasoning and graduate-level domain-specific reasoning. For testing, we select 50 problems each from MATH500 and GPQA and use the entire AIME24 dataset. we set $\mathcal{S} = \mathcal{W} = 512$, $\tau = 0.9$ and the maximum generation length to 16K, and apply stochastic sampling with a temperature of 0.6 and a top-p value of 0.95, following the original DeepSeek-R1 setup. Since the outputs of reasoning models are highly sensitive to random seeds [41], we generate $k = 8$ different samples for each problem. We report two metrics: **pass@**k, which measures the likelihood of at least one correct solution among the k samples, and **avg@**k, which represents the average accuracy across all k samples.

As shown in Table 3, FreeKV delivers accuracy comparable to models with full KV cache and outperforms other compression methods across most datasets. KV dropping methods such as RazorAttention and RaaS exhibit significant accuracy losses, particularly on AIME24, which involves more complex problems. Moreover, FreeKV consistently outperforms other KV retrieval methods in most cases, demonstrating the effectiveness of its page summaries, softmax-based group consistent selection, and fine-grained correction mechanism.

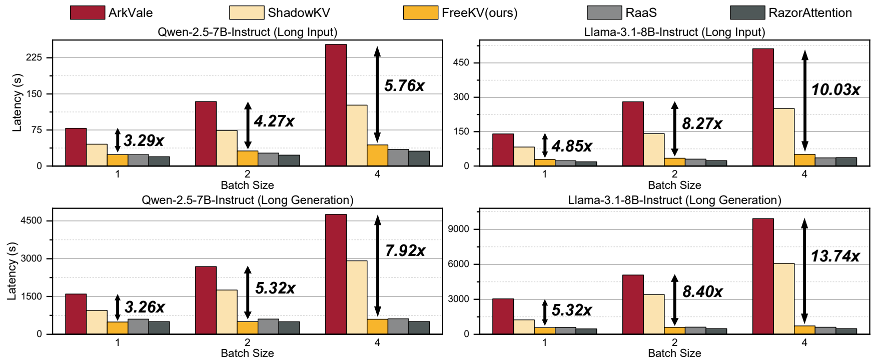


Figure 7: End-to-end latency comparison between KV compression methods.

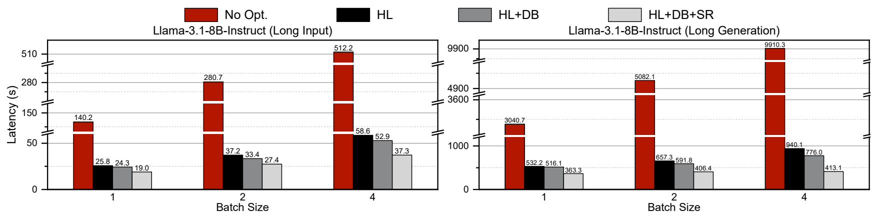


Figure 8: Ablation results for efficiency optimizations.

5.3 Efficiency evaluation

Setup Our experiments were conducted on an Nvidia A100 40GB GPU, connected with AMD 7302 CPUs via PCIe Gen4. The evaluation covers Qwen-2.5-7B and Llama-3.1-8B models under both long-input (32K input, 512 output) and long-generation scenarios (600 input, 16K output). We set τ to 0.8 for long-input and 0.9 for long-generation scenarios, with $\mathcal{B} = 2048$ and $\mathcal{S} = \mathcal{W} = 512$.

End-to-end latency As shown in Fig. 7, FreeKV demonstrate significant efficiency gains over SoTA KV retrieval methods, achieving up to 13.7× and 8.4× speedups compared to ArkVale and ShadowKV, respectively. Moreover, FreeKV attains efficiency comparable to dropping methods like RaaS and RazorAttention, which do not involve offloading or recall. The speedups over ArkVale are detailed in Fig. 7, whereas the long-input and long-generation speedups over ShadowKV with a batch size of 4 are 2.9× and 4.9× for Qwen-2.5-7B-Instruct, and 5× and 8.4× for Llama-3.1-8B-Instruct. The improvements become more pronounced for large batch sizes and in long-generation scenarios, where more recall operations are required. In addition, the improvements are amplified for Llama-3.1-8B, which with more KV heads and a larger KV cache compared to Qwen-2.5-7B.

Ablation study We present the ablation results of efficiency optimizations applied in FreeKV, including hybrid layouts (HL), double-buffering streamed recall (DB) and speculative retrieval (SR), evaluated using Llama-3.1-8B-Instruct under long-input and long-generation scenarios. As shown in Fig. 8, hybrid layouts, which eliminate fragmented data transfers, contribute the most to the improvements, achieving up to a 10.5× speedup. For a batch size of 4, streamed recall adds a further 1.2× speedup, while overlapping with speculative retrieval provides an additional 1.9× speedup.

6 Discussion

While FreeKV achieves near-lossless accuracy, techniques such as adaptive budgets [42] or dynamic budgets with top-p sparsity [20, 43, 44] can be applied orthogonally to further enhance accuracy. In addition, machine learning based methods have been proposed to predict attention patterns for KV cache compression [45, 46]. However, these methods introduce significant training and runtime overhead and are only effective for long-input scenarios. Moreover, although page-wise selection is found to be less effective for small budgets [21, 47], learnable block-wise sparsity techniques, applied during pre-training [31, 48] or post-training [49], show promise in achieving native and optimal page-wise KV cache compression and retrieval.

7 Conclusion

We present FreeKV, an algorithm-system co-optimization KV retrieval framework that integrates speculative retrieval and fine-grained correction on the algorithm side, as well as hybrid layouts and streamed recall on the system side. FreeKV achieves near-lossless accuracy across various scenarios and models, delivering up to a 13× speedup over SOTA KV retrieval methods.

References

[1] Robert Chew, John Bollenbacher, Michael Wenger, Jessica Speer, and Annice Kim. Llm-assisted content analysis: Using large language models to support deductive coding, 2023.

[2] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

[3] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation, 2024.

[4] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[5] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[6] Google DeepMind. Gemini 2.5: Our most intelligent ai model, March 2025.

[7] xAI. Grok 3 beta — the age of reasoning agents, February 2025.

[8] Meta. The llama 3 herd of models, 2024.

[9] Yao Fu. Challenges in deploying long-context transformers: A theoretical peak performance analysis, 2024.

[10] Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*, 2025.

[11] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *ICLR*, 2024.

[12] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads, 2024.

[13] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024.

[14] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710. Curran Associates, Inc., 2023.

[15] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation, 2024.

[16] Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. Efficient long-decoding inference with reasoning-aware attention sparsity, 2025.

[17] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *ICML*, 2024.

[18] Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative llm inference with recallable key-value eviction. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 113134–113155. Curran Associates, Inc., 2024.

[19] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference, 2025.

[20] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation, 2024.

[21] Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression, 2024.

[22] Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. Rethinking key-value cache compression techniques for large language model serving, 2025.

[23] Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. Can llms maintain fundamental abilities under kv cache compression?, 2025.

[24] OpenAI. Openai o1 system card. 2024.

[25] Qwen. Qwq-32b: Embracing the power of reinforcement learning, March 2025.

[26] Greg Kamradt. Needle in a haystack - pressure testing llms, 2023.

[27] 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. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[28] Huggingface. Maxwell-jia/aime_2024, February 2025.

[29] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore, December 2023. Association for Computational Linguistics.

[30] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[31] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025.

[32] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024.

[33] Huaijin Wu, Lianqiang Li, Hantao Huang, Tu Yi, Jihang Zhang, Minghui Yu, and Junchi Yan. HShare: Fast LLM decoding by hierarchical key-value sharing. In *The Thirteenth International Conference on Learning Representations*, 2025.

[34] flashinfer ai. Kv-cache layout in flashinfer, 2025.

[35] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[36] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*, 2024.

[37] Yuhao Wu, Ming Shan Hee, Zhiqing Hu, and Roy Ka-Wei Lee. Longgenbench: Benchmarking long-form generation in long context llms, 2024.

[38] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[39] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

[40] Qwen. Qwen3: Think deeper, act faster, May 2025.

[41] Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility, 2025.

[42] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference, 2024.

[43] Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning, 2025.

[44] Qihui Zhou, Peiqi Yin, Pengfei Zuo, and James Cheng. Progressive sparse attention: Algorithm and system co-design for efficient attention in llm serving, 2025.

[45] Qingyue Yang, Jie Wang, Xing Li, Zhihai Wang, Chen Chen, Lei Chen, Xianzhi Yu, Wulong Liu, Jianye Hao, Mingxuan Yuan, and Bin Li. Attentionpredictor: Temporal pattern matters for efficient llm inference, 2025.

[46] Yash Akhauri, Ahmed F AbouElhamayed, Yifei Gao, Chi-Chih Chang, Nilesh Jain, and Mohamed S. Abdelfattah. Tokenbutler: Token importance is predictable, 2025.

[47] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference, 2024.

[48] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025.

[49] Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. Seerattention: Learning intrinsic sparse attention in your llms, 2025.