SPARK: Query-Aware Unstructured Sparsity with Recoverable KV Cache Channel Pruning

Huanxuan Liao[1, 2*†], Yixing Xu[3*], Shizhu He[1, 2‡], Guanchen Li[3], Xuanwu Yin[3], Dong Li[3], Emad Barsoum[3], Jun Zhao[1, 2], Kang Liu[1, 2]

[1] The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, China
[2] School of Artificial Intelligence, University of Chinese Academy of Sciences
[3] Advanced Micro Devices, Inc., Beijing, China
liaohuanxuan2023@ia.ac.cn

Abstract

Long-context inference in large language models (LLMs) is increasingly constrained by the KV cache bottleneck: memory usage grows linearly with sequence length, while attention computation scales quadratically. Existing approaches address this issue by compressing the KV cache along the *temporal* axis through strategies such as token eviction or merging to reduce memory and computational overhead. However, these methods often neglect fine-grained importance variations across feature dimensions (i.e., the *channel axis*), thereby limiting their ability to effectively balance efficiency and model accuracy. In reality, we observe that channel saliency varies dramatically across both queries and positions: certain feature channels carry near-zero information for a given query, while others spike in relevance. To address this oversight, we propose SPARK, a training-free plug-and-play method that applies unstructured sparsity by pruning KV at the channel level, while dynamically restoring the pruned entries during attention score computation. Notably, our approach is orthogonal to existing KV compression and quantization techniques, making it compatible for integration with them to achieve further acceleration. By reducing channel-level redundancy, SPARK enables processing of longer sequences within the same memory budget. For sequences of equal length, SPARK not only preserves or improves model accuracy but also reduces KV cache storage by over 30% compared to eviction-based methods. Furthermore, even in an aggressive pruning ratio of 80%, SPARK maintains performance with less degradation than 5% compared to the based eviction method, demonstrating robustness and effectiveness.

Code — https://github.com/AMD-AGI/AMD-Spark
Extended version — https://arxiv.org/abs/2508.15212

1 Introduction

Large language models (LLMs) are increasingly deployed in diverse and complex tasks requiring extended (even infinite) contextual understanding (Liu et al. 2025; Tan et al. 2025), such as book summarization (Kim et al. 2024), instruction following (Liao et al. 2024) and code or math reasoning (Liao et al. 2025b). To support these applications, recent models like GPT-4 (Achiam et al. 2023), Gemini-2.5 (Comanici et al. 2025), and Qwen-3 (Yang et al. 2025) have scaled to 100K+ token contexts. However, handling such long sequences poses serious challenges in memory and latency due to the growing Key-Value (KV) cache in inference (Tang et al. 2024b). For example, storing the KV cache for 100K tokens in LLaMA3.1-8B (Dubey et al. 2024) exceeds $50GB$, surpassing the model size itself (Shutova et al. 2025; Liao et al. 2025a). For a hidden size of 128, matrix multiplication latency increases from $2ms$ at $1K$ tokens to $764ms$ at 16K, nearly $380\times$ slower. Consequently, KV cache has become a critical bottleneck, restricting the scalability and deployment of LLMs in long-context scenarios (Fu 2024).

Specifically, the total KV cache size is determined by the batch size B, sequence length S, number of layers L, attention heads N, and the head dimension D. Prior efforts on KV cache compression have primarily targeted the following aspects: 1) **Temporal axis (S)**: by evicting (Ge et al. 2023; Zhang et al. 2023) or merging (Wan et al. 2024; Wang et al. 2024) unimportant tokens using attention scores or redundancy heuristics (Cai et al. 2025). 2) **Spatial axis (L, N)**: by sharing KV across similar layers (Brandon et al. 2024; Wu and Tu 2024) or pruning attention heads with limited contribution to long-range dependencies (Xiao et al. 2025). 3) **Channel axis (D)**: by applying low-rank decomposition (Liu et al. 2024a; Sun et al. 2024a) or structured pruning (Xu

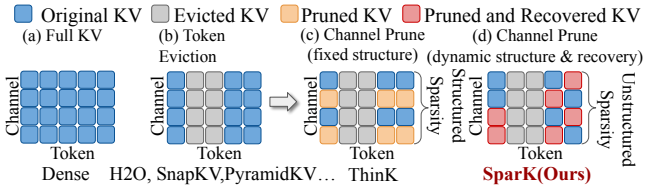


Figure 1: **Illustrative comparisons** among (a) full KV cache, (b) eviction-based KV compression, (c) structured channel pruning-based KV reduction, and (d) our proposed SPARK, which employs unstructured channel pruning with subsequent recovery during attention score computation.

*Equal Contribution.
†Work done during an internship at AMD.
‡Corresponding author.

et al. 2024). 4) **Quantization**: by applying low-bit precision storage (Hooper et al. 2024b; Zhang et al. 2025).

However, these approaches predominantly adopt structured channel sparsity, applying uniform pruning strategies that either discard or retain entire channels, or enforce fixed pruning masks across all tokens (Shi et al. 2024). Such methods rest on the assumption that channel importance remains consistent throughout the input sequence, which overlooks the dynamic and token-specific nature of attention in LLMs. Moreover, by applying identical pruning masks to both keys and queries, these methods fail to account for the asymmetric roles and token-wise variability in channel saliency, ultimately limiting the flexibility of the dot-product attention mechanism. Instead of directly discarding unimportant channels, we argue that *replacing unimportant channel entries with approximate or low-magnitude entries* can mitigate attention score distortion and maintain performance even under an aggressive pruning ratio.

In this paper, we propose SPARK, a method that introduces fine-grained query-aware unstructured sparsity to the KV cache while guaranteeing the recoverability of pruned channel entries. We reformulate channel pruning as a **critical channel set** selection problem aimed at maximizing aggregate saliency across selected channels. To this end, we introduce a lightweight metric to quantify the per-token, per-channel importance and adopt a greedy algorithm to solve the resulting optimization problem efficiently (Bi et al. 2024). To mitigate information loss under high pruning ratios, we further introduce a recovery mechanism that approximates the contributions of pruned channels through a recovery function $\mathcal{F}$ during attention computation. This approximation ensures effective information retention without incurring additional memory cost. We additionally explore value cache pruning via a simple norm-based heuristic, showing promising results and paving the way for future refinement. Furthermore, we propose two ratio-free variants: group-based (SPARK-g) and top-p pruning (SPARK-p), demonstrating the flexibility and generality of SPARK.

Extensive experimental evaluations demonstrate the effectiveness of SPARK across a wide range of scenarios, benchmarks (Bai et al. 2024; Hsieh et al. 2024; Hao et al. 2025b), and LLMs (Dubey et al. 2024; Yang et al. 2025). Importantly, SPARK is compatible with prior methods that optimize S, L and N. When integrated with token eviction strategies, SPARK not only preserves computational efficiency and achieves comparable or superior accuracy but also reduces KV cache storage by over 30%. Remarkably, even at high channel pruning ratio ($\geq 70\%$) while maintaining the same sequence length via token eviction methods such as SnapKV (Li et al. 2024) or PyramidKV (Yang et al. 2024a), SPARK maintains performance degradation within 5% compared to the based method, significantly outperforming THINK, which incurs a 47.6% accuracy loss under similar settings. Our main contributions are listed as follows:

- We propose SPARK, a novel training-free plug-and-play KV cache compression approach that introduces unstructured fine-grained sparsity along the channel dimension. We reformulate the pruning task as a critical channel set selection problem that aims to maximize the saliency contribution of preserved channels.
- We introduce an on-the-fly recovery mechanism that approximates the contribution of pruned channels during attention score computation using a lightweight function $\mathcal{F}$ to mitigate information loss with little increasing memory footprint or computational overhead.
- Extensive experiments show that our method consistently achieves remarkable effectiveness in various benchmarks and LLM. Notably, even when pruning 80% of the channels at the same sequence length, the performance degradation remains within 5%.

2 Related Work

Existing KV cache compression methods can be broadly categorized into three categories based on dimensions: **temporal-axis**, **spatial-axis**, and **channel-axis** methods.

Temporal-Axis Optimization reduces the sequence length S to alleviate the linear memory growth in long-context inference (Liao et al. 2025d; Liu et al. 2024b). *Token eviction* methods selectively remove low-contributing tokens based on attention scores (Li et al. 2024; Ge et al. 2023; Yang et al. 2024a; Liao et al. 2025c) or redundancy heuristics (Cai et al. 2025). *Token merging* techniques compress inputs by merging semantically similar tokens (Nawrot et al. 2024; Wan et al. 2024; Wang et al. 2024; Hao et al. 2025a) or aggregating discarded ones (Hooper et al. 2024a; Zhang et al. 2024). Paged KV cache architectures, such as vLLM (Kwon et al. 2023), further enhance scalability via memory paging.

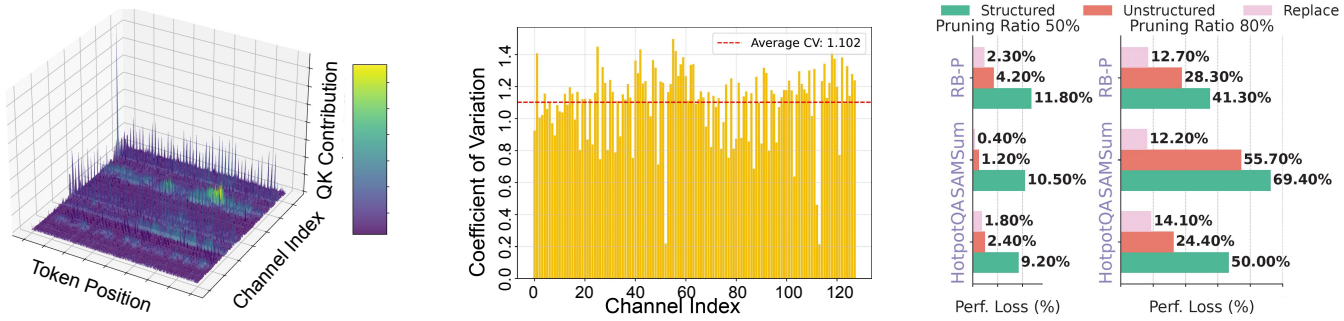
Spatial-Axis Optimization reduces redundancy by shrinking the number of layers L or heads N. Cross-layer sharing (Sun et al. 2024b; Yang et al. 2024b) enables KV reuse across layers, while MQA (Shazeer 2019) and GQA (Ainslie et al. 2023) share KV pairs across heads. Head optimization aims to prune attention heads that are less sensitive to long-range dependencies (Fu et al. 2024; Tang et al. 2024a; Zhu et al. 2024), and DuoAttention (Xiao et al. 2025) specializes heads for retrieval or streaming to enhance efficiency.

Channel-Axis Optimization targets the channel dimension D to reduce KV cache memory. Low-rank methods (Sun et al. 2024a) decompose KV matrices into compact representations, while MLA (Liu et al. 2024a) learns latent heads to compress channels, requiring retraining. Closest to our work, THINK (Xu et al. 2024) performs query-guided structured pruning, but its structured strategy significantly degrades performance under high pruning ratios. In contrast, we propose unstructured, dynamic pruning with on-the-fly recovery, enabling adaptive removal and restoration of KV entries during computation.

3 Preliminaries

LLM inference comprises two stages (Liu et al. 2025): **prefill** and **decode**. During prefill, the entire input sequence is processed in parallel to generate the first output token. Given a prompt embedding $\mathbf{X} \in \mathbb{R}^{S \times H}$, where S is the sequence length and H is the hidden dimension, the key and value matrices for each attention head $i \in [1, N]$ are computed as:

$$\boldsymbol{\mathcal{K}}_i = \mathbf{X}\mathbf{W}_k^i, \quad \boldsymbol{\mathcal{V}}_i = \mathbf{X}\mathbf{W}_v^i, \tag{1}$$



(a) **3D surface visualization** of score contributions, highlighting token-wise unstructured sparsity.

(b) **Coefficient of Variation** (CV) distribution across channels, suggesting dynamic channel modulation for different contexts.

(c) **Metric comparison**: performance loss caused by the method, highlighting the importance of unstructured pruning and recovery.

Figure 2: Rethinking the salience of key channels using LLaMA3.1-8B-Instruct (Dubey et al. 2024) on Longbench (Bai et al. 2024). All visualizations are derived from the 18th attention layer and the 0th attention head.

where $\mathbf{W}_k^i, \mathbf{W}_v^i \in \mathbb{R}^{H \times D}$ are the projection matrices for the i-th head, and D is the dimensionality of each head. The resulting keys and values are stored in the KV cache. During decode, each newly generated token embedding $\mathbf{x} \in \mathbb{R}^{1 \times H}$ is projected to obtain the corresponding query, key, and value vectors and appended to the existing KV cache:

$$\mathbf{q}_i = \mathbf{x}\mathbf{W}_q^i, \quad \mathbf{k}_i = \mathbf{x}\mathbf{W}_k^i, \quad \mathbf{v}_i = \mathbf{x}\mathbf{W}_v^i. \tag{2}$$

$$\boldsymbol{\mathcal{K}}_i \leftarrow \text{Cat}[\boldsymbol{\mathcal{K}}_i, \mathbf{k}_i], \quad \boldsymbol{\mathcal{V}}_i \leftarrow \text{Cat}[\boldsymbol{\mathcal{V}}_i, \mathbf{v}_i]. \tag{3}$$

The attention output for each head is then computed as:

$$\mathbf{a}_i = \text{Softmax}\left(\frac{\mathbf{q}_i \boldsymbol{\mathcal{K}}_i^\top}{\sqrt{D}}\right), \quad \mathbf{o}_i = \mathbf{a}_i \boldsymbol{\mathcal{V}}_i. \tag{4}$$

Finally, the outputs $\mathbf{o}_i$ from all heads are concatenated and passed to the feed-forward network (FFN). In scenarios involving extended contexts or large batch processing, the primary bottlenecks in memory consumption and computational speed stem from the KV size. While existing approaches primarily focus on reducing KV size through temporal (S) or spatial (L, N) optimization, we draw inspiration from THINK (Xu et al. 2024) and propose optimizing the KV cache from channel D, thereby offering a complementary and orthogonal direction for KV compression.

4 Methodology

In this section, we begin with an experimental analysis and motivation for SPARK in Sec.4.1, followed by problem formulation and analysis in Sec.4.2. We further introduce the proposed SPARK in Sec.4.3.

4.1 Motivations and Observations

To understand the role of individual key channels, we conduct an empirical analysis[1] of the QK dot-product scores. As shown in Figure 2, we observe **unstructured, token-dependent channel** importance patterns that vary significantly across different tokens, which motivates the need for adaptive pruning strategies that can dynamically select different channels for different tokens, rather than applying uniform pruning across the entire sequence (Jie et al. 2025).

Observation 1: Token-wise Unstructured Channel Sparsity. Empirical analysis reveals that attention heads exhibit highly unstructured channel-wise sparsity, varying significantly across tokens. As shown in Figure 2(a), the 3D surface visualization highlights token-dependent activation patterns, where different tokens rely on distinct subsets of channels. This contradicts structured pruning assumptions where importance is globally consistent. To quantify this variability, we compute the coefficient of variation (CV) across tokens for each channel, as illustrated in Figure 2(b). The average CV exceeds 1.1, indicating that token-wise fluctuations dominate. This suggests that channel importance is highly context sensitive and cannot be accurately captured through a static and structured sparsity. Figure 2(c) further demonstrates that unstructured pruning, which respects token-level heterogeneity, substantially outperforms structured pruning. At 50% pruning, unstructured pruning leads to only 1.2% performance drop (vs. 4.2% for structured); at 80% pruning, it maintains a 27.4% gap (28.3% vs. 55.7%). These results affirm the necessity of unstructured sparsity.

Observation 2: Retaining Dimensional Structure Mitigates Pruning Impact. Figure 2(c) also shows that replacing pruned channel entries with minimal constant values (e.g., 0.01) during attention score computation rather than zeroing or omitting them yields substantial performance gains. This lightweight strategy preserves the structural integrity of the attention mechanism while avoiding pruning queries. Under 80% pruning, this approach significantly narrows the performance gap. On SAMSum, it reduces performance degradation from 55.7% to 12.2%; on HotpotQA, from 69.4% to 41.3%; and on RB-P, from 50.0% to 24.4%. On average, the substitution of entries reduces the loss of accuracy by 32. 4% compared to removal. These results highlight that even a coarse query-agnostic constant of pruning channel can play a pivotal role in maintaining performance.

[1]More analysis and metric details refer to the Appendix B.

4.2 Problem Formulation and Analysis

Let $\mathcal{C}_{i,t} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_D\}$ denote the original channel set for each head i and token t, where D is the head dimension. We aim to select a subset $\hat{\mathcal{C}}_{i,t} \subseteq \mathcal{C}_{i,t}$ of T channels ($T \ll D$) that retain the most salient attention contributions, thereby enhancing inference efficiency while minimizing performance degradation. To formalize this, we introduce a binary mask $\mathcal{S}_{i,t} = \{\boldsymbol{z}^1_{i,t}, \ldots, \boldsymbol{z}^D_{i,t}\} \in \{0,1\}^D$ with $\boldsymbol{z}^j_{i,t} \in \{0,1\}$ indicating whether channel j is retained ($\boldsymbol{z}^j_{i,t} = 1$) or pruned ($\boldsymbol{z}^j_{i,t} = 0$). Our primary goal is to *minimize* the discrepancy ($\mathcal{E}$) in attention weights after pruning:

$$\min_{\mathcal{S}_{i,t}} \mathcal{E}(\mathcal{S}_{i,t}) = \left\| \mathbf{q}_{i,t}\mathbf{k}^\top_{i,t} - (\mathbf{q}_{i,t}\mathcal{S}_{i,t})(\mathbf{k}_{i,t}\mathcal{S}_{i,t})^\top \right\|_F, \quad (5)$$

where $\|\|_F$ denotes the Frobenius norm for vectors. Solving this combinatorial problem exactly is intractable as it corresponds to a cardinality-constrained low-rank approximation. To derive an approximate solution, we expand the squared Frobenius norm of $\mathcal{E}$ for each token t:

$$\begin{aligned} \mathcal{E}(\mathcal{S}_{i,t})^2 \ =& \sum_{j=1}^{D} \|\mathbf{q}^j_{i,t}\|_2^2 \|\mathbf{k}^j_{i,t}\|_2^2 (1 - \boldsymbol{z}^j_{i,t})^2 + \\ & 2 \sum_{\substack{j,r=1 \\ j<r}}^{D} \langle \mathbf{q}^j_{i,t}, \mathbf{q}^r_{i,t} \rangle \langle \mathbf{k}^j_{i,t}, \mathbf{k}^r_{i,t} \rangle (1 - \boldsymbol{z}^j_{i,t}\boldsymbol{z}^r_{i,t}), \end{aligned} \quad (6)$$

where $\mathbf{q}^j_{i,t}$ and $\mathbf{k}^j_{i,t}$ are the j-th dimensions of $\mathbf{q}_{i,t}$ and $\mathbf{k}_{i,t}$ respectively (similarly for r). The first term measures individual contributions of each pruned channel, while the second reflects inter-channel redundancy. In practice, we observe that different channels are nearly uncorrelated (i.e., $\langle \mathbf{k}^j_{i,t}, \mathbf{k}^r_{i,t} \rangle \approx 0$ for $j \neq r$), allowing us to drop the second term. Thus, minimizing $\mathcal{E}(\mathcal{S}_{i,t})$ is well-approximated by *minimizing* the sum of the norms of pruned channel contributions for each token, which is equivalent to *maximizing* retained channel scores while the number of selected channels for each token is fixed: $\sum_j^D \boldsymbol{z}^j_{i,t} = T$. We introduce a proxy saliency score $\boldsymbol{w}^j_{i,t} = \|\mathbf{q}^j_{i,t}\|_2 \|\mathbf{k}^j_{i,t}\|_2$, which upper bounds the contribution of channel j at token t to the Frobenius norm. The optimization problem is reformulated as follows:

$$\max_{\boldsymbol{z}^j_{i,t}} \sum_{j=1}^{D} \boldsymbol{w}^j_{i,t} \boldsymbol{z}^j_{i,t} \quad \text{s.t.} \quad \sum_{j=1}^{D} \boldsymbol{z}^j_{i,t} = T, \quad \forall t, \quad (7)$$

Since the objective is linear and additive in $\boldsymbol{z}_j$, the optimal solution is simply to select the T channels with the highest saliency score $\boldsymbol{w}_j$, which can be efficiently solved using a greedy algorithm: $\hat{\mathcal{C}}_{i,t} = \text{Top}_T(\boldsymbol{w}^1_{i,t}, \ldots, \boldsymbol{w}^D_{i,t})$. Given the pruning ratio λ, we only keep the $T = \lfloor (1-\lambda)D \rfloor$ most important channels among D channels of each head.

4.3 SPARK

Building on above analysis, we redefine the channel pruning problem as (Eq. 6). Since this study focuses on efficiency in long-context inference, we employ a heuristic algorithm with relatively low computational complexity to obtain an approximate solution. To this end, we introduce an unstructured channel pruning method (Figure 3), which selects an appropriate channel subset while ensuring that it satisfies the constraint. Our approach is training-free, plug-and-play, and model-agnostic, which makes it applicable to any LLM.

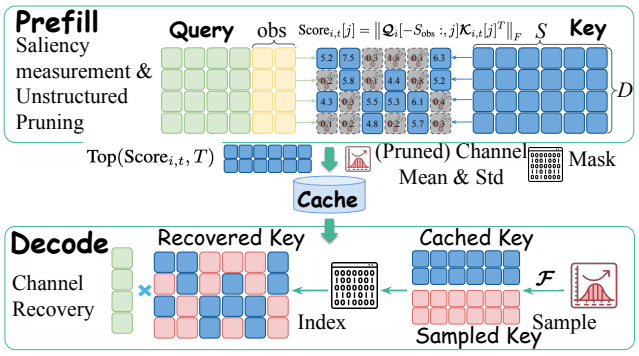


Figure 3: **An illustration of SPARK.** SPARK computes channel-wise saliency scores and applies unstructured pruning during prefill. During decoding, SPARK leverage $\mathcal{F}$ and sampling from the cached distribution to reconstruct the pruned channels and then perform standard full attention.

As illustrated in Figure 3, the proposed method consists of two primary phases: 1) unstructured channel pruning based on saliency measurement during **prefill**, and 2) channel recovery using stored distribution patterns during **decode**. Following previous work (Li et al. 2024; Xu et al. 2024), to reduce the computation cost, we only use the last observation window to calculate the saliency score. Specifically, we approximate the attention interaction by replacing per-token query vectors with the mean query vector computed over a local observation window. Specifically, for an observation window of size W, the mean query vector $\overline{\mathbf{q}}_i$ for the head i is calculated as the average of the query vectors $\mathbf{q}_{i,t}$ over the window: $\overline{\mathbf{q}}^j_i = \frac{1}{W} \sum_{t=t_0}^{t_0+W-1} \mathbf{q}^j_{i,t}$, where t_0 is the starting token index of the window.

Saliency Measurement and Unstructured Pruning. We compute the proxy saliency score $\boldsymbol{w}_{j,t}$ for each channel j and token t to estimate per-channel contribution to the attention mechanism. We sort the scores in descending order and construct a binary pruning mask $\mathcal{S}_i \in \{0,1\}^{S \times D}$ for head i, retaining the top-T channels. The pruned key matrix is denoted as $\hat{\boldsymbol{\mathcal{K}}}_i = \boldsymbol{\mathcal{K}}_i[\mathcal{S}_i] \in \mathbb{R}^{S \times T}$, where $\boldsymbol{\mathcal{K}}_i[\mathcal{S}_i]$ extracts the channels indexed by $\mathcal{S}$. To support recovery during decoding, we further compute the distributional statistics[2] (mean μ_i, standard deviation σ_i) of the saliency scores, or the mean of pruned entries $\mu_{i,\text{pruned}}$. These statistics are critical for recovering approximations of the pruned channels as our goal is to select channels with lower final attention scores, rather than those with inherently small key entries, given the non-trivial dependency of scores on query key interactions.

Channel Recovery. Based on Observation 2 in Section 4.1, we propose a *query-aware recovery function* $\mathcal{F}$ to recon-

[2]Detailed formulations are provided in Appendix A.2.

	Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
		NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
-	Vanilla	22.48	44.72	46.23	48.49	44.71	24.43	30.7	22.8	27.28	72.0	88.35	42.28	6.5	72.0	63.61	51.67	44.27
KV-size 128	StreamingLLM	13.64	18.03	17.79	31.36	27.46	8.67	17.31	18.99	17.87	31.0	31.21	35.71	1.5	67.5	56.63	55.16	28.11
	ExpectedAttention	17.32	24.08	23.87	38.76	26.43	12.55	22.26	20.81	23.57	20.5	77.22	36.59	5.5	62.5	52.78	46.45	31.95
	TOVA	17.09	23.35	37.88	43.32	28.68	15.85	19.87	20.54	18.51	26.5	85.18	39.15	4.0	60.5	59.98	57.17	34.85
	SnapKV	15.29	20.03	29.2	39.92	28.26	15.06	17.74	19.27	18.05	21.0	68.64	36.64	6.0	66.0	57.86	59.08	32.38
	+THINK (0.5)	13.6	19.2	31.78	36.24	25.05	11.92	16.85	19.17	16.4	2.0	50.73	32.8	6.0	65.0	52.29	52.11	28.20
	+THINK (0.8)	7.6	7.03	17.45	19.98	9.8	6.9	14.37	14.15	12.5	0.0	21.36	11.22	1.02	63.0	30.42	34.69	16.97
	+**SPARK** (0.5)	13.52	20.19	29.28	38.77	26.33	14.44	17.66	19.12	17.98	21.0	68.95	36.66	5.5	65.5	58.49	59.18	32.04
	+**SPARK** (0.8)	13.82	20.28	28.63	40.84	26.75	14.25	17.29	19.06	17.23	22.0	57.2	35.41	7.0	64.0	57.2	57.61	31.16
	PyramidKV	21.79	44.6	45.96	48.33	43.63	25.82	30.42	22.45	27.05	72.0	88.69	41.59	6.0	71.5	62.21	48.72	43.80
	+THINK (0.5)	22.48	40.56	47.94	45.83	34.95	23.19	27.55	22.54	25.73	53.5	84.88	32.7	7.78	71.0	53.9	51.54	40.38
	+THINK (0.8)	6.37	5.53	13.73	12.53	5.47	3.16	16.97	14.21	17.02	0.0	23.03	7.54	1.73	13.0	29.67	27.51	12.34
	+**SPARK** (0.5)	22.66	43.95	45.82	48.33	43.85	24.85	30.16	22.76	26.84	70.0	88.34	41.4	6.5	71.5	62.83	51.15	43.81
	+**SPARK** (0.8)	22.44	44.2	44.62	46.29	40.37	22.68	27.83	22.56	25.67	69.0	84.2	40.17	5.5	72.0	60.38	41.98	41.87
KV-size 512	StreamingLLM	13.98	23.72	20.26	35.82	29.76	11.34	22.12	19.56	24.49	45.0	54.98	38.32	4.5	67.0	58.16	52.63	32.6
	ExpectedAttention	19.73	33.41	30.2	45.06	32.81	20.43	25.55	21.45	26.25	51.0	85.76	39.57	6.0	56.0	62.0	54.84	38.13
	TOVA	18.84	33.46	44.0	48.36	36.82	21.47	23.07	20.72	24.33	63.0	88.91	41.01	6.0	71.0	64.66	58.33	41.5
	SnapKV	19.24	36.51	43.61	46.83	36.62	23.11	22.62	21.17	24.03	45.0	88.59	40.09	6.0	71.5	63.75	58.65	40.46
	+THINK (0.5)	18.73	33.83	41.47	43.72	27.98	20.91	20.59	21.56	22.25	15.5	84.62	33.82	7.0	71.5	57.01	56.97	36.09
	+THINK (0.8)	9.48	6.59	18.62	18.28	8.32	9.2	17.11	15.37	16.46	0.0	43.94	8.6	2.21	34.62	33.43	35.47	17.36
	+**SPARK** (0.5)	18.66	36.13	43.23	46.66	36.17	22.86	22.44	21.19	23.7	42.5	89.11	40.15	6.5	71.5	63.8	59.0	40.22
	+**SPARK** (0.8)	18.23	37.34	42.42	44.71	34.85	23.14	21.8	21.26	23.68	41.5	87.22	38.88	5.0	72.5	62.86	55.01	39.40
	PyramidKV	21.79	44.6	45.96	48.33	43.63	25.82	30.42	22.45	26.96	72.0	88.69	41.59	6.0	71.5	62.21	48.72	43.79
	+THINK (0.5)	22.48	40.56	47.94	45.83	34.95	23.19	27.55	22.54	25.6	53.5	84.88	32.7	7.78	71.0	53.9	51.54	40.37
	+THINK (0.8)	6.37	5.53	13.73	12.53	5.47	3.16	16.97	14.21	17.11	0.0	23.03	7.54	1.73	13.0	29.67	27.51	12.35
	+**SPARK** (0.5)	22.79	43.99	45.63	48.83	43.64	24.87	30.34	22.89	26.57	70.0	88.75	42.28	6.5	71.5	62.72	50.81	43.88
	+**SPARK** (0.8)	22.73	44.1	47.2	46.47	40.51	22.81	26.66	22.72	24.87	68.0	88.63	40.44	5.5	72.0	59.61	42.44	42.17

Table 1: Performance comparison on LLaMA-3-8B-Instruct at LongBench. **SPARK** (λ) and THINK(λ) denote the channel-wise key cache pruning ratio λ. Full results including other cache budgets and additional models are provided in Appendix F.2.

struct pruned key channels, addressing the limitations of discard or fixed-value replacement. We utilize cached distributional statistics collected during the prefill stage to sample plausible score values and then back-compute the corresponding key entries. Specifically, we sample a score $\tilde{\boldsymbol{w}}_{j,t}$ and the sampled key entry is computed as $\tilde{\mathbf{k}}_{i,t}^{j} = \frac{\tilde{\boldsymbol{w}}_{i,t}^{j}}{\|\overline{\mathbf{q}}_i^j\|_2}$, ensuring that the inner product $\langle \overline{\mathbf{q}}_i^j, \tilde{\mathbf{k}}_{i,t}^{j} \rangle \approx \tilde{\boldsymbol{w}}_{i,t}^{j}$, consistent with the sampled score. We consider the following instantiations of the recovery function $\boldsymbol{\mathcal{F}}$:

- **Gaussian distribution:** $\tilde{\boldsymbol{w}}_{i,t}^{j} \sim \mathcal{N}(\mu_i, \sigma_i^2)$
- **Exponential distribution:** $\tilde{\boldsymbol{w}}_{i,t}^{j} \sim \mathrm{Exp}(1/\mu_i)$
- **Degenerate (only μ) distribution:** $\tilde{\boldsymbol{w}}_{i,t}^{j} = \mu_{i,\text{pruned}}$

The choice of distribution is flexible and can be configured per head or globally. Empirically, degenerate sampling performs robustly across tasks and layers. Overall, the $\boldsymbol{\mathcal{F}}$ is defined as:

$$\tilde{\mathbf{k}}_{i,t}^{j} = \boldsymbol{\mathcal{F}}(\mu, \sigma) = \frac{\text{sample}(\text{dist}(\mu, \sigma))}{\|\overline{\mathbf{q}}_i^j\|_2}, \tag{8}$$

Finally, we reconstruct the full key matrix $\tilde{\boldsymbol{\mathcal{K}}}_i$ by combining the cached pruned keys with the sampled keys according to the mask $\mathcal{S}_i$, ensuring both structural completeness and numerical consistency of the attention computation.

5 Experiments

5.1 Experimental Setup

Benchmark Datasets. We evaluate our SPARK against state-of-the-art KV cache compression methods on three widely recognized long-context understanding benchmarks: LongBench (Bai et al. 2024) and RULER (Hsieh et al. 2024) to thoroughly assess SPARK's achievable performance.

Implementation Details. To validate SPARK's general effectiveness, we evaluate on LLMs of varying scales and capabilities, including LLaMA-3/3.1-8/70B-Instruct (Dubey et al. 2024), Qwen3-8B/32B (Yang et al. 2025). To ensure a fair comparison between KV cache compression strategies and their integration with SPARK, we adopt consistent hyperparameter settings across all settings. Unless otherwise specified, we apply SPARK to the *key cache* only and use the *degenerate distribution* as the default recovery strategy.

Baselines. We benchmark SPARK against the standard full KV cache and prior KV cache compression methods, including StreamingLLM (Xiao et al. 2023), PyramidKV (Yang et al. 2024a), SnapKV (Li et al. 2024) and ExpectedAttention (Jegou et al. 2024) under various cache budgets.

Method	Niah1	Niah2	Niah3	MKey1	MKey2	MKey3	MValue	MQuery	VT	CWE	FWE	QA1	QA2	Avg.
Vanilla	100.0	100.0	100.0	99.6	100	99.2	99.1	99.0	99.8	88.9	90.0	81.0	57.2	93.36
StreamingLLM	18.8	17.4	19.0	20.2	20.0	18.4	18.25	18.2	32.84	0.18	81.33	31.4	33.6	25.35
ExpectedAttention	99.2	42.0	3.4	33.8	57.0	0.8	9.35	21.1	66.12	54.46	70.6	72.0	48.2	44.46
TOVA	100.0	100.0	97.8	99.4	96.8	0.4	98.9	99.25	99.76	54.04	90.8	77.4	54.6	82.24
SnapKV	100.0	100.0	10.0	99.8	97.2	63.2	97.7	99.45	97.36	53.92	85.73	80.8	57.2	80.18
+THINK(0.5)	96.6	99.6	9.4	99.0	92.2	55.4	98.55	98.25	94.84	29.12	88.87	76.0	50.6	76.03
+THINK(0.8)	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.0	0.0	0.32	0.0	18.8	20.2	3.03
+**SPARK**(0.5)	100.0	100.0	10.2	99.4	96.6	62.8	98.05	99.45	97.64	53.8	86.2	80.8	56.0	80.07
+**SPARK**(0.8)	100.0	99.8	9.6	99.2	94.2	49.4	98.1	98.75	96.64	41.12	87.07	80.0	53.8	77.51
PyramidKV	100.0	100.0	5.0	99.8	98.2	55.0	98.6	99.35	98.6	16.88	87.0	80.0	57.2	76.59
+THINK(0.5)	97.2	100.0	4.8	99.4	93.0	49.2	98.7	98.75	96.16	8.46	88.33	76.2	52.4	74.05
+THINK(0.8)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.24	0.0	14.8	19.4	2.65
+**SPARK**(0.5)	99.2	99.2	5.2	99.4	97.6	54.4	97.95	98.7	98.16	16.84	86.27	79.6	56.8	76.1
+**SPARK**(0.8)	99.4	98.8	5.2	99.2	94.4	44.2	97.1	97.7	95.24	12.08	86.2	78.4	54.0	73.99

Table 2: RULER evaluation results on the LLaMA3.1-8B-Instruct model with SPARK under a 20% KV cache budget and 16K input length. Additional results across varying cache budgets and input lengths are reported in Appendix F.3 for completeness.

Additional experimental details can refer to Appendix C.

5.2 Benchmark on LongBench

Table 1 presents the performance comparison of KV compression methods and their integration with our proposed key cache channel pruning for LLaMA-3-8B-Instruct, evaluated in various KV budgets on the LongBench. The pruning ratio $\lambda = 0.8$ indicates that 80% of key cache channels are removed, resulting in a 40% reduction in the total KV cache memory. The following observations can be drawn:

Compatibility with Existing Methods. When integrated with token eviction strategies (e.g., PyramidKV), SPARK further boosts effectiveness. Comparisons between SnapKV and PyramidKV integrated with channel pruning further validate the robustness and general applicability of SPARK. Notably, the stronger the eviction strategy, the greater the gains observed from incorporating SPARK. Combining SPARK(0.5) outperforms the integrated eviction baseline and combining SPARK(0.8) maintains 95% of accuracy while reducing cache storage by 40%.

Superior Performance under High Pruning Ratios. SPARK consistently outperforms THINK across all budgets and pruning ratios. In particular, under a high pruning ratio ($\lambda = 0.8$), we observe that integrating THINK with either SnapKV or PyramidKV leads to substantial degradation in performance (average drop of **65%**). In contrast, combining **SPARK** with the same baselines incurs less than **5%** average performance loss. SPARK's recoverable pruning preserves both expressivity and stability even at 80% sparsity, while THINK suffers catastrophic degradation.

5.3 Benchmark on RULER

Table 2 presents the results of RULER under 20% cache budget. SPARK consistently outperforms THINK while preserving competitive accuracy under all settings. Notably, under a stringent cache budget (20% or 50%) with 8K and 16K inputs, THINK (0.8) suffers drastic degradation with performance dropping below 3%, while SPARK (0.8) retains accuracy within 3% of baseline eviction methods, highlighting the effectiveness of our recovery mechanism. Even at moderate pruning (e.g., 0.5), SPARK consistently outperforms THINK and matches or surpasses baseline strategies, demonstrating both accuracy preservation and general applicability of our method SPARK.

5.4 Analysis

We conduct a comprehensive evaluation of SPARK across three key dimensions: pruning ratio, input length, and cache size. Results are summarized in Figure 4.

Impact of Pruning Ratio. Figure 4(a) shows that SPARK consistently outperforms THINK and the unrecovered variant, particularly under high compression. At $\lambda = 0.8$, THINK incurs a performance drop exceeding 35%, whereas SPARK maintains a degradation within 5%. This highlights the effectiveness of channel-aware pruning and query-aware recovery in preserving attention quality.

Throughput under Long Inputs. Figure 4(b) illustrates the decoding throughput across varying input lengths with KV budget of 128. While the full-cache baseline fails beyond 64k due to memory overflow, SPARK sustains high throughput across all lengths. Notably, SPARK achieves comparable throughput to THINK, despite the added recovery step. This indicates that the recovery mechanism introduces negligible overhead in decoding latency.

Cache Size vs. Performance. As shown in Figure 4(c), SPARK achieves superior performance under the same or smaller cache budgets. By pruning key channels, both SPARK and THINK achieve lower memory usage than SnapKV under the same KV size. Compared to THINK, SPARK consistently delivers performance closer to SnapKV across varying compression ratios. Under equal memory budgets, SPARK outperforms all baselines, underscoring its effectiveness in complementing KV compression methods for improved memory efficiency.

Pruning Value Cache Channels. We further extend SPARK to support simultaneous pruning of both key and value cache channels (λ_k + λ_v) in the Appendix D. As shown in Table 5, SPARK maintains strong robustness under joint prun-

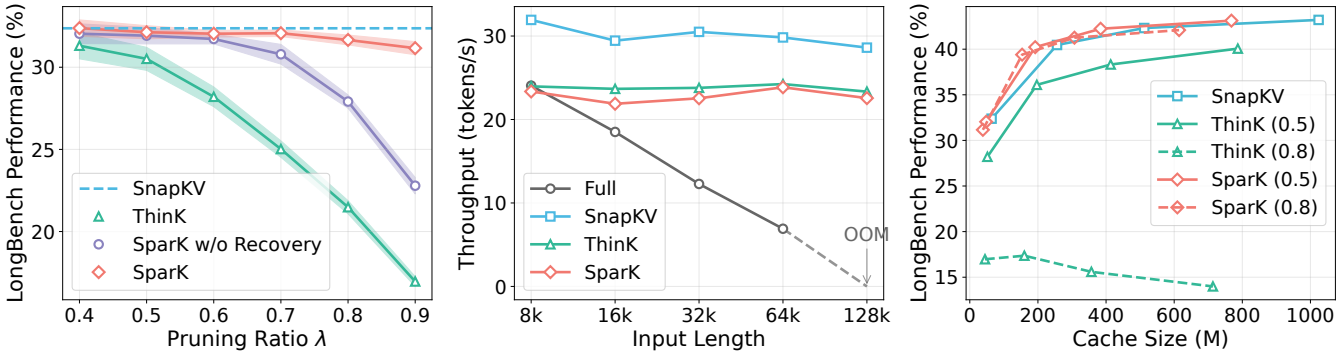


Figure 4: Performance–Efficiency analysis of SPARK on LLaMA3-8B-Instruct. (a) LongBench average performance under varying pruning ratios (λ). SPARK significantly outperforms THINK across all compression levels. (b) Throughput (tokens/s) with increasing input length. SPARK maintains stable decoding speed across long sequences (up to 128k) (c) Cache size vs. performance trade-off. SPARK achieves favorable efficiency–performance balance compared to THINK and SnapKV.

Dist.	$\lambda = 0.5$				$\lambda = 0.8$			
	128	**512**	**1024**	**2048**	**128**	**512**	**1024**	**2048**
Norm.	**32.71**	39.76	41.37	42.18	31.25	38.78	41.03	41.68
Exp.	32.56	40.16	42.18	43.04	**31.43**	39.04	41.21	41.87
Deg.	32.04	**40.22**	**42.24**	**43.13**	31.16	**39.41**	**41.26**	**42.09**

Table 3: Ablation study on recovery distribution.

Variants	Pruning Ratio (λ)	Threshold (p)	Group (g)	KV-Size				Overall Ratio
				128	**512**	**1024**	**2048**	
SPARK	0.5	-	-	32.04	40.22	42.24	43.13	0.50
SPARK-p	-	99%	-	32.11	40.13	42.18	42.95	0.58
SPARK-g	-	-	5	32.06	40.17	42.17	42.76	0.55
SPARK-g	-	-	4	32.11	40.11	42.45	43.27	0.44

Table 4: Ablation study on variants.

ing. For example, under the (0.5+0.5) configuration with SnapKV in 128 KV-size, the average performance drops marginally from 32.38 to 32.03. Interestingly, both (0.5+0.3) and (0.5+0.5) achieve performance comparable to or exceeding the single-stage (0.5) baseline. Although extreme compression (0.8+0.8) incurs more noticeable degradation, the recovery mechanism limits the additional loss to within 5% on average. These results show that SPARK generalizes effectively to joint KV pruning, delivering substantial memory savings under moderate settings while preserving accuracy, and underscore both the flexibility of channel-wise sparsity and the essential role of recovery.

5.5 Ablation Studies

Unless stated otherwise, all ablation experiments are conducted on the LongBench benchmark using the LLaMA3-8B-Instruct model with various KV budgets.

Recovery Distributions. We investigate the impact of different recovery distributions under two pruning ratios ($\lambda = 0.5$ and 0.8). As shown in Table 3, all achieve comparable performance, indicating that SPARK is largely insensitive to the choice of statistical model. Degenerate recovery attains the best results on long inputs, reflecting greater stability under aggressive pruning. Gaussian and Exponential provide moderate flexibility but may introduce marginal noise that does not consistently benefit attention approximation when keys are highly constrained. The Exponential distribution performs slightly better at shorter sequences, likely due to its heavier tail increasing diversity in sampled keys.

Adaptive Variants of SPARK. We further explore two adaptive variants of SPARK designed to eliminate the need for a manually-tuned pruning ratio. The first, SPARK-p, employs dynamic thresholding, retaining the most salient channels whose cumulative importance scores exceed a threshold of 99%. The second, SPARK-g, partitions the D channels into g disjoint groups ranked by importance and applies progressively aggressive pruning to less salient groups. For our experiments, we set the pruning ratios to $(0.25, 0.5, 0.75, 1.0)$ for $g = 4$ and $(0.1, 0.3, 0.5, 0.7, 0.9)$ for $g = 5$. As shown in Table 4, both variants match the fixed-ratio baseline's performance. Notably, the grouped variant with $g = 4$ achieves the highest performance with a lower average pruning ratio (0.44), suggesting that fine-grained sparsity can lead to better trade-offs between compression and performance. These results validate SPARK's flexibility and the effectiveness of its adaptive extensions.

6 Conclusion

In this paper, we introduce SPARK, a novel channel-wise pruning that leverages unstructured sparsity alongside a lightweight statistical recovery mechanism. Unlike prior methods that suffer from significant degradation under high pruning ratios, SPARK preserves attention fidelity by selectively retaining salient channels and reconstructing pruned entries using cached statistics. Extensive experiments demonstrate that SPARK significantly reduces memory consumption and maintains competitive performance, highlighting the importance of channel recovery in mitigating the adverse effects of aggressive pruning.

Acknowledgements

This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDA04080400) and Beijing Natural Science Foundation (L243006) and the National Natural Science Foundation of China (No.62376270).

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ainslie, J.; Lee-Thorp, J.; De Jong, M.; Zemlyanskiy, Y.; Lebrón, F.; and Sanghai, S. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; et al. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3119–3137.

Bi, Z.; Han, K.; Liu, C.; Tang, Y.; and Wang, Y. 2024. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*.

Brandon, W.; Mishra, M.; Nrusimha, A.; Panda, R.; and Ragan-Kelley, J. 2024. Reducing transformer key-value cache size with cross-layer attention. *Advances in Neural Information Processing Systems*, 37: 86927–86957.

Cai, Z.; Xiao, W.; Sun, H.; Luo, C.; Zhang, Y.; Wan, K.; Li, Y.; Zhou, Y.; Chang, L.-W.; Gu, J.; et al. 2025. R-KV: Redundancy-aware KV Cache Compression for Training-Free Reasoning Models Acceleration. *arXiv preprint arXiv:2505.24133*.

Comanici, G.; Bieber, E.; Schaekermann, M.; Pasupat, I.; Sachdeva, N.; Dhillon, I.; Blistein, M.; Ram, O.; Zhang, D.; et al. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. arXiv:2507.06261.

Dao, T. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; Goyal, A.; Hartshorn, A. S.; Yang, A.; Mitra, A.; Sravankumar, A.; Korenev, A.; Hinsvark, A.; Rao, A.; Zhang, A.; and et al. 2024. The Llama 3 Herd of Models. *ArXiv*, abs/2407.21783.

Fu, Y. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv preprint arXiv:2405.08944*.

Fu, Y.; Cai, Z.; Asi, A.; Xiong, W.; Dong, Y.; and Xiao, W. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*.

Ge, S.; Zhang, Y.; Liu, L.; Zhang, M.; Han, J.; and Gao, J. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Hao, Y.; Cao, P.; Jin, Z.; Liao, H.; Chen, Y.; Liu, K.; and Zhao, J. 2025a. CITI: Enhancing Tool Utilizing Ability in Large Language Models without Sacrificing General Performance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 23996–24004.

Hao, Y.; Cao, P.; Jin, Z.; Liao, H.; Chen, Y.; Liu, K.; and Zhao, J. 2025b. Evaluating personalized tool-augmented llms from the perspectives of personalization and proactivity. *arXiv preprint arXiv:2503.00771*.

Hooper, C.; Kim, S.; Mohammadzadeh, H.; Maheswaran, M.; Paik, J.; Mahoney, M. W.; Keutzer, K.; and Gholami, A. 2024a. Squeezed attention: Accelerating long context length llm inference. *arXiv preprint arXiv:2411.09688*.

Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M. W.; Shao, Y. S.; Keutzer, K.; and Gholami, A. 2024b. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37: 1270–1303.

Hsieh, C.-P.; Sun, S.; Kriman, S.; Acharya, S.; Rekesh, D.; Jia, F.; Zhang, Y.; and Ginsburg, B. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654*.

Jegou, S.; Jeblick, M.; Austin, D.; and Devoto, A. 2024. kvpress.

Jie, S.; Tang, Y.; Han, K.; Deng, Z.-H.; and Han, J. 2025. SpeCache: Speculative Key-Value Caching for Efficient Generation of LLMs. *arXiv preprint arXiv:2503.16163*.

Kim, Y.; Chang, Y.; Karpinska, M.; Garimella, A.; Manjunatha, V.; Lo, K.; Goyal, T.; and Iyyer, M. 2024. Fables: Evaluating faithfulness and content selection in book-length summarization. *arXiv preprint arXiv:2404.01261*.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J. E.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37: 22947–22970.

Liao, H.; He, S.; Hao, Y.; Li, X.; Zhang, Y.; Zhao, J.; and Liu, K. 2025a. SKIntern: Internalizing Symbolic Knowledge for Distilling Better CoT Capabilities into Small Language Models. In *Proceedings of the 31st International Conference on Computational Linguistics*, 3203–3221.

Liao, H.; He, S.; Xu, Y.; Zhang, Y.; Hao, Y.; Liu, S.; Liu, K.; and Zhao, J. 2024. From instance training to instruction learning: Task adapters generation from instructions. *Advances in Neural Information Processing Systems*, 37: 45552–45577.

Liao, H.; He, S.; Xu, Y.; Zhang, Y.; Liu, K.; and Zhao, J. 2025b. Neural-symbolic collaborative distillation: Advancing small language models for complex reasoning tasks. In

Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 24567–24575.

Liao, H.; He, S.; Xu, Y.; Zhang, Y.; Liu, S.; Liu, K.; and Zhao, J. 2025c. Awakening Augmented Generation: Learning to Awaken Internal Knowledge of Large Language Models for Question Answering. In *Proceedings of the 31st International Conference on Computational Linguistics*, 1333–1352.

Liao, H.; Hu, W.; Xu, Y.; He, S.; Zhao, J.; and Liu, K. 2025d. Beyond Hard and Soft: Hybrid Context Compression for Balancing Local and Global Information Retention. *arXiv preprint arXiv:2505.15774*.

Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Dengr, C.; Ruan, C.; Dai, D.; Guo, D.; et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Liu, J.; Tang, D.; Huang, Y.; Zhang, L.; Zeng, X.; Li, D.; Lu, M.; Peng, J.; Wang, Y.; Jiang, F.; et al. 2024b. Updp: A unified progressive depth pruner for cnn and vision transformer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 13891–13899.

Liu, J.; Zhu, D.; Bai, Z.; He, Y.; Liao, H.; Que, H.; Wang, Z.; Zhang, C.; Zhang, G.; Zhang, J.; et al. 2025. A Comprehensive Survey on Long Context Language Modeling. *arXiv preprint arXiv:2503.17407*.

Nawrot, P.; Łańcucki, A.; Chochowski, M.; Tarjan, D.; and Ponti, E. M. 2024. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*.

Paszke, A. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

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

Shi, L.; Zhang, H.; Yao, Y.; Li, Z.; and Zhao, H. 2024. Keep the cost down: A review on methods to optimize LLM's KV-cache consumption. *arXiv preprint arXiv:2407.18003*.

Shutova, A.; Malinovskii, V.; Egiazarian, V.; Kuznedelev, D.; Mazur, D.; Surkov, N.; Ermakov, I.; and Alistarh, D. 2025. Cache Me If You Must: Adaptive Key-Value Quantization for Large Language Models. *arXiv preprint arXiv:2501.19392*.

Sun, H.; Chang, L.-W.; Bao, W.; Zheng, S.; Zheng, N.; Liu, X.; Dong, H.; Chi, Y.; and Chen, B. 2024a. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*.

Sun, Y.; Dong, L.; Zhu, Y.; Huang, S.; Wang, W.; Ma, S.; Zhang, Q.; Wang, J.; and Wei, F. 2024b. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37: 7339–7361.

Tan, Y.; He, S.; Liao, H.; Zhao, J.; and Liu, K. 2025. Dynamic parametric retrieval augmented generation for test-time knowledge enhancement. *arXiv preprint arXiv:2503.23895*.

Tang, H.; Lin, Y.; Lin, J.; Han, Q.; Hong, S.; Yao, Y.; and Wang, G. 2024a. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*.

Tang, J.; Zhao, Y.; Zhu, K.; Xiao, G.; Kasikci, B.; and Han, S. 2024b. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*.

Wan, Z.; Wu, X.; Zhang, Y.; Xin, Y.; Tao, C.; Zhu, Z.; Wang, X.; Luo, S.; Xiong, J.; and Zhang, M. 2024. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*.

Wang, Z.; Jin, B.; Yu, Z.; and Zhang, M. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Wu, H.; and Tu, K. 2024. Layer-condensed kv cache for efficient inference of large language models. *arXiv preprint arXiv:2405.10637*.

Xiao, G.; Tang, J.; Zuo, J.; Yang, S.; Tang, H.; Fu, Y.; Han, S.; et al. 2025. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *The Thirteenth International Conference on Learning Representations*.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Xu, Y.; Jie, Z.; Dong, H.; Wang, L.; Lu, X.; Zhou, A.; Saha, A.; Xiong, C.; and Sahoo, D. 2024. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, D.; Han, X.; Gao, Y.; Hu, Y.; Zhang, S.; and Zhao, H. 2024a. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. In *Findings of the Association for Computational Linguistics ACL 2024*, 3258–3270.

Yang, Y.; Cao, Z.; Chen, Q.; Qin, L.; Yang, D.; Zhao, H.; and Chen, Z. 2024b. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*.

Zhang, Y.; Du, Y.; Luo, G.; Zhong, Y.; Zhang, Z.; Liu, S.; and Ji, R. 2024. Cam: Cache merging for memory-efficient llms inference. In *Forty-first international conference on machine learning*.

Zhang, Z.; Gao, Y.; Fan, J.; Zhao, Z.; Yang, Y.; and Yan, S. 2025. SelectQ: Calibration Data Selection for Post-training Quantization. *Machine Intelligence Research*, 22: 499–510.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36: 34661–34710.

Zhu, H.; Tang, D.; Liu, J.; Lu, M.; Zheng, J.; Peng, J.; Li, D.; Wang, Y.; Jiang, F.; Tian, L.; et al. 2024. Dip-go: A diffusion pruner via few-step gradient optimization. *Advances in Neural Information Processing Systems*, 37: 92581–92604.