ResKV: Residual-based Channel-wise Unstructured Pruning for KV Cache Compression

Yue Chen, Jinze Li, Dajiang Liu*
College of Computer Science, Chongqing University, Chongqing, 401331, China
Jialingjiang Laboratory, Chongqing, 401331, China
Jialingjiang Laboratory, Chongqing University, Chongqing, 401331, China

Abstract

In modern LLM applications, the demand for extended context lengths continues to surge, making KV cache a major performance bottleneck in long-context inference. Channel-wise pruning has emerged as a prevalent KV cache compression paradigm; however, existing solutions overlook inherent similarities among KV vectors, resulting in limited pruning ratios and noticeable performance degradation. To tackle this issue, we present ResKV, a novel residual-driven channel-wise pruning framework that fully exploits KV vector similarity for effective compression. ResKV first clusters highly similar KV vectors via cosine similarity metrics, then calculates residual vectors by subtracting the corresponding cluster centroid from the original features. Subsequent channel-wise unstructured pruning and per-token quantization are exclusively applied to these residual components. During inference, preserved residuals are combined with their corresponding centroids to accurately recover dense KV representations. Benefiting from the low magnitude and sparse, near-zero properties of centroid residuals, both pruning and quantization incur only negligible approximation errors. Experimental results show that, at 75% sparsity, ResKV achieves 4.27× the inference throughput of the dense baseline while sustaining competitive model performance. Under the same sparsity setting, ResKV improves the average LongBench score by up to 2.26 points over state-of-the-art channel-wise KV pruning methods and achieves 2.63× the end-to-end throughput of a competing pruning baseline. Our code is publicly available at `https://github.com/coralabo/ResKV`.

1 Introduction

Large language models (LLMs) have empowered a wide range of applications, such as document analysis, search engines, math reasoning, etc. To enable these complex tasks, the demand for long context length is rapidly increasing. For example, recent models like GPT-4 OpenAI et al. [2024] and Qwen-3 Yang et al. [2025] have scaled to 100K+ tokens context. However, as the size of Key and Value (KV) cache grows linearly with the context length, enabling such long contexts poses serious challenges in both memory footprint and access latency Pope et al. [2023], even leading to inference failure Wang et al. [2024], Sheng et al. [2023]. As a result, the KV cache has become a critical bottleneck when deploying LLMs in long-context scenarios.

To mitigate the high overhead of KV cache, extensive research efforts have been devoted to KV cache compression techniques, including token-wise selection/eviction Zhang et al. [2023], Tang et al. [2024], Li et al. [2024], Zhang et al. [2025a], quantization Liu et al. [2024a], He et al. [2024], Hooper et al. [2024], and channel-wise pruning Xu et al. [2025], Joo et al. [2025], Zhang et al. [2025b]. For channel-wise pruning, less significant channel subsets of each token are permanently

*Corresponding author. Email: `liudj@cqu.edu.cn`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

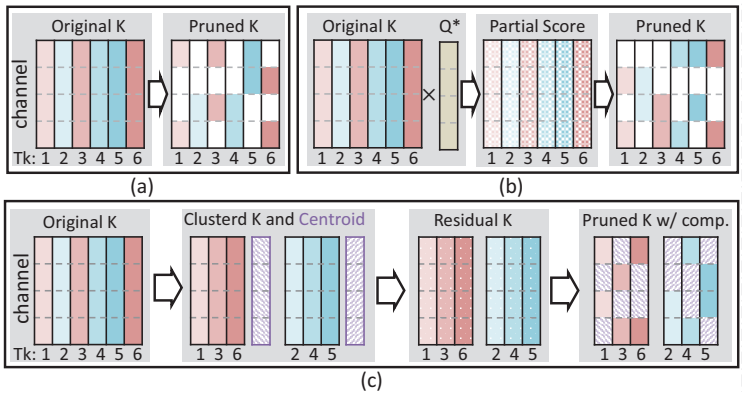


Figure 1: Comparison of different channel-wise pruning methods. (a) Input-based pruning (Mustafar), (b) output-based pruning (SparK), and (c) residual-based pruning (ResKV).

pruned from the feature space. Based on the significance metrics adopted for channel selection, existing channel-wise pruning methods can be categorized into two main types: input-based pruning and output-based pruning. For input-based pruning methods Joo et al. [2025], as illustrated in Fig. 1(a), unstructured pruning is directly applied to the input key vectors according to the magnitude of their elements. In contrast, output-based pruning methods Xu et al. [2025], Liao et al. [2025], as shown in Fig. 1(b), first compute the dot product of all key vectors with a set of observed query (Q^*) vectors to generate an output partial score matrix, after which structured or unstructured pruning is performed on the key vectors based on the matrix values. While both input-based and output-based pruning can reduce the KV cache footprint to a certain extent, they suffer from two critical limitations. First, they fail to achieve a favorable trade-off between sparsity and inference accuracy. For instance, when the pruning-induced sparsity reaches 87.5%, the resulting inference accuracy loss exceeds 20%. Second, these methods are incompatible with subsequent quantization operations: channel-wise quantization can scarcely leverage the benefits of unstructured pruning, whereas token-wise quantization is severely affected by the large magnitudes of the retained elements in pruned vectors. Collectively, these inherent limitations render current channel-wise pruning approaches far from optimal for KV cache compression.

To address these limitations, we turn to the well-documented semantic similarity of KV vectors and leverage this inherent property to optimize channel-wise KV pruning. Extensive prior work has identified notable semantic similarity among KV vectors Liu et al. [2025], Hu et al. [2025], Sun et al. [2025], an observation that has been widely harnessed to advance KV cache quantization and token eviction/selection strategies. *However, existing channel-wise KV pruning methods overlook this critical semantic similarity, thus missing a key opportunity to refine the sparsity-accuracy trade-off for KV cache compression.*

We therefore introduce ResKV, a residual-based channel-wise pruning approach that fully incorporates KV vector semantic similarity into the pruning pipeline to mitigate the above drawbacks. As illustrated in Fig. 1(c), ResKV first groups semantically similar KV vectors into clusters and computes representative centroids. It then transforms each KV vector into a low-magnitude residual by subtracting the corresponding cluster centroid, and applies channel-wise unstructured pruning only on these residuals. During inference, ResKV uses differentiated reconstruction strategies for Keys and Values to efficiently restore dense KV representations with minimal accuracy loss and computational overhead. Our main contributions are summarized as follows:

- We present ResKV, a plug-and-play KV cache compression method utilizing KV semantic similarity for channel-wise unstructured pruning. A novel centroid residual metric guides pruning to alleviate accuracy drops, delivering a better sparsity-accuracy trade-off than existing channel pruning methods.

- We realize an organic integration of unstructured pruning and quantization for KV cache. By conducting post-pruning quantization on residual vectors, low-bit quantization incurs nearly no accuracy loss in LLM inference.

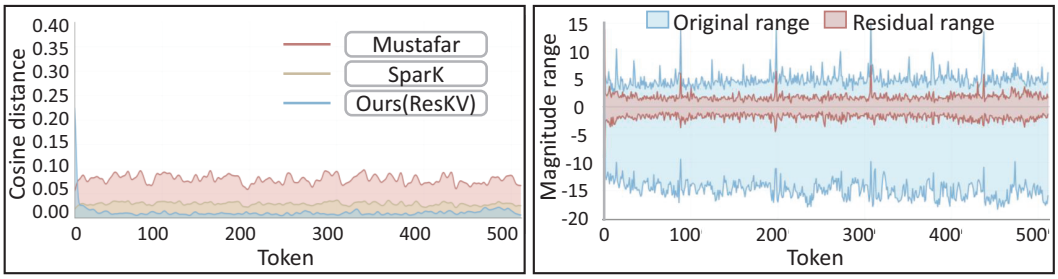


Figure 2: Motivation analysis on Llama-3-8B-Instruct at 75% sparsity using WikiText-2 (Layer 16, Head 1). (a) The cosine distance between original Key vector and pruned Key vector of different approaches. (b) The feature magnitude of the original Key vector and the residual Key vector.

- We achieve seamless integration of ResKV with standard Flash Decoding Dao et al. [2023], realizing a lightweight and drop-in enhancement. By decoupling Key and Value processing, ResKV maintains high accuracy while moderately reducing computational overhead.

2 Background

LLM inference proceeds in two phases: a parallel *prefill* phase, followed by sequential *autoregressive decoding* Agrawal et al. [2023]. For a specific attention head at decoding step t, the hidden state $\mathbf{h}_t \in \mathbb{R}^{1\times D_{\text{model}}}$ is projected and modulated by Rotary Positional Embeddings (RoPE) Su et al. [2021], denoted as $\mathcal{R}(\cdot)$, to produce the query and key vectors:

$$\mathbf{q}_t = \mathcal{R}(\mathbf{h}_t\mathbf{W}_{\text{Q}}), \quad \mathbf{k}_t = \mathbf{h}_t\mathbf{W}_{\text{K}}, \quad \hat{\mathbf{k}}_t = \mathcal{R}(\mathbf{h}_t\mathbf{W}_{\text{K}}), \quad \mathbf{v}_t = \mathbf{h}_t\mathbf{W}_{\text{V}} \tag{1}$$

where $\mathbf{W}_{\{\text{Q,K,V}\}} \in \mathbb{R}^{D_{\text{model}}\times D_h}$ are the projection weights. To avoid redundant computation, the inference engine usually maintains KV cache, appending the current key and value vectors to the history cache $(\widehat{\mathbf{K}}_t, \mathbf{V}_t)$:

$$\widehat{\mathbf{K}}_t \leftarrow [\widehat{\mathbf{K}_{t-1}}; \hat{\mathbf{k}}_t], \quad \mathbf{V}_t \leftarrow [\mathbf{V}_{t-1}; \mathbf{v}_t] \tag{2}$$

The output $\mathbf{o}_t$ is then derived by attending to the entire cached sequence:

$$\mathbf{o}_t = \text{softmax}\left(\frac{\mathbf{q}_t\widehat{\mathbf{K}_t}^{\top}}{\sqrt{D_h}}\right)\mathbf{V}_t \tag{3}$$

While the prefill phase is *compute-bound*, the decoding phase is strictly *memory-bound*. It relies on general matrix-vector (GEMV) operations that require streaming the continuously growing KV cache $(\widehat{\mathbf{K}}_t, \mathbf{V}_t)$ from High Bandwidth Memory (HBM) for every token, resulting in low arithmetic intensity and significant memory access overhead.

3 Motivation

To understand how clustering affects channel-wise pruning and per-token quantization, we conduct an empirical analysis of the cosine distance between the original key vector and the pruned key vector (with centroid compensation) and the feature magnitude of the original Key vector and residual Key vector. Detailed visualization analysis for the Value cache is provided in Appendix A. Based on the analysis, we obtain two observations.

Observation 1: Input/output-based pruning incurs a larger cosine distance. The cosine distance between the original vector and the pruned vector is widely used to evaluate the impact of pruning. If the pruned vector has smaller cosine distance from the original vector, it is more likely to generate higher inference accuracy. Therefore, as shown in Fig. 2(a), we present the cosine distance between the original key vector and the pruned vector (with 75% pruning rate) of Mustafar, SparK and our method, employing compensation for pruned elements where applicable. The results show that residual-based pruning in ResKV achieves the lowest cosine distance, with an average value of

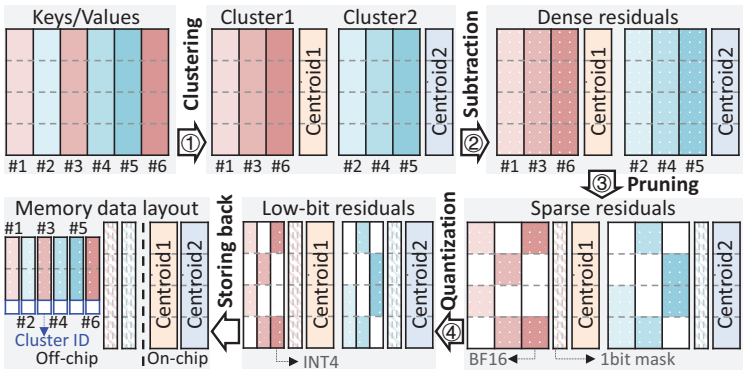


Figure 3: The compression process of ResKV.

0.01. We note a localized increase at initial tokens. This is attributed to the model's disproportionate sensitivity to these positions Xiao et al. [2024], Liu et al. [2024b], which induces feature outliers that are challenging for centroid approximation. In contrast, input-based pruning in Mustafar and output-based pruning in SparK can only achieve average cosine distances of 0.08 and 0.05, respectively. Thus, with a larger cosine distance, the input-based pruning and output-based pruning are prone to severe accuracy degradation with the same pruning rate or require lower sparsity to maintain comparable inference accuracy.

Observation 2: Input/output-based unstructured pruning offers limited compatibility with per-token quantization. When performing channel-wise unstructured pruning, it is challenging to implement per-channel quantization like KIVI Liu et al. [2024a] as adjacent tokens may have different pruned channels. Therefore, channel-wise unstructured pruning prefers per-token quantization and the magnitudes of features within a token's KV vectors are crucial to low-bit quantization. Unfortunately, many works Liu et al. [2024a]Hooper et al. [2024] find that the key cache exhibits distinct channel-wise outliers, which can hardly be pruned in channel-wise pruning. Fig. 2(b) depicts the feature magnitude ranges of the original Key vector and the residual Key vector. Notably, the magnitude ranges of Mustafar and SparK are nearly identical to those of the original key vector. We observe that the pruned key vectors in Mustafar and SparK exhibit an identical average magnitude of 19.77, which is consistent with the original magnitude range. In contrast, the residual vector in our approach only has an average magnitude of 3.69. Thus, without any reduction in the magnitude range, both input-based and output-based pruning schemes are incompatible with per-token quantization—an approach that is virtually the sole viable option for unstructured channel-wise pruning.

4 Method

In this section, we first introduce the residual-based KV compression scheme of ResKV. Then, we present the optimized Flash Decoding workflow with differentiated Key and Value residual processing. Finally, we describe the full inference pipeline for both prefill and decoding stages.

4.1 Residual-Guided KV Compression

As discussed earlier, ResKV leverages low-magnitude residuals for efficient channel-wise pruning and per-token quantization. A key insight is that Key features exhibit strong semantic similarity before positional encoding, which is significantly distorted after RoPE. In contrast to existing methods that compress Keys after RoPE (Equation (2)), ResKV stores raw pre-RoPE KV vectors ($\mathbf{K}_t \leftarrow [\mathbf{K}_{t-1}; \mathbf{k}_t]$) and performs clustering and compression directly on them.

In addition, Key states present limited local similarity as token distance grows, while Value features naturally lack obvious spatial locality. To address this limitation, we abandon local window clustering and adopt global K-means to aggregate semantically correlated KV pairs, producing low-amplitude residuals relative to cluster centroids.

Fig. 3 illustrates the end-to-end compression pipeline of ResKV. Given the raw KV features before RoPE injection, we first partition all cached Key and Value vectors into N_C clusters using K-means,

Algorithm 1: ResKV based Flash Decoding

Input : $\widehat{\mathbf{q}}$, RoPE-applied query vector; $\mathbf{R}_K, \mathbf{R}_V$, Sparse residual KV; $\mathbf{B}_K, \mathbf{B}_V$, 128-bit bitmap; $\mathbf{I}_K, \mathbf{I}_V$, Cluster ID per token; $\mathbf{C}_K, \mathbf{C}_V$, Cluster centroids; N_C, Cluster number

Output : Attention output $\mathbf{O}$

// Parallel processing over KV chunks

parallel for *each chunk* s **do**

  acc_val $\leftarrow \mathbf{0}^{128}$, **acc_cluster** $\leftarrow \mathbf{0}^{N_C}$, acc_exp $\leftarrow 0$, current_max $\leftarrow -\infty$

  // Iterate each KV token in the chunk

  for *each token* t *in chunk* s **do**

    // 1. Restore full Key and compute score

    $i_k \leftarrow \mathbf{I}_K[t]$

    $\mathbf{k} \leftarrow \texttt{Restore}(\mathbf{R}_K[t], \mathbf{B}_K[t]) + \mathbf{C}_K[i_k]$

    $\mathbf{k} \leftarrow \texttt{RoPE}(\mathbf{k}, t)$

    score $\leftarrow \widehat{\mathbf{q}}^\top \cdot \mathbf{k}/\sqrt{D_h}$

    // 2. Online softmax

    new_max $\leftarrow \max($current_max, score$)$

    old_scale $\leftarrow \exp($current_max $-$ new_max$)$

    new_weight $\leftarrow \exp($score $-$ new_max$)$

    // 3. Restore residual value only

    $i_v \leftarrow \mathbf{I}_V[t]$

    $\mathbf{v} \leftarrow \texttt{Restore}(\mathbf{R}_V[t], \mathbf{B}_V[t])$

    // 4. Accumulate values and cluster weights

    acc_val $\leftarrow$ **acc_val** $\cdot$ old_scale $+$ new_weight $\cdot \mathbf{v}$

    acc_cluster $\leftarrow$ **acc_cluster** $\cdot$ old_scale

    acc_cluster$[i_v] \leftarrow$ **acc_cluster**$[i_v] +$ new_weight

    acc_exp $\leftarrow$ acc_exp $\cdot$ old_scale $+$ new_weight

    current_max $\leftarrow$ new_max

  end

  SaveChunk(**acc_val**, **acc_cluster**, acc_exp, current_max)

end

// Global chunk merging and centroid fusion

parallel for *each batch, head* **do**

  (**total_val**, **total_cluster**, total_exp) $\leftarrow$ MergeChunk()

  total_val $\leftarrow$ **total_val** $+$ **total_cluster** $\cdot \mathbf{C}_V$

  $\mathbf{O} \leftarrow$ **total_val**/total_exp

end

where each cluster is represented by a centroid $\mathbf{C}_K$ and $\mathbf{C}_V$ in BF16 format. For each token t, we compute its *residual vectors* by subtracting the corresponding cluster centroid indexed by cluster ID $\mathbf{I}_K[t]$ and $\mathbf{I}_V[t]$:

$$\mathbf{R}_K[t] = \mathbf{K}_t - \mathbf{C}_K[\mathbf{I}_K[t]], \quad \mathbf{R}_V[t] = \mathbf{V}_t - \mathbf{C}_V[\mathbf{I}_V[t]]. \tag{4}$$

Subsequently, we perform channel-wise unstructured pruning on the residual vectors $\mathbf{R}_K$ and $\mathbf{R}_V$, generating compact 128-bit binary masks $\mathbf{B}_K$ and $\mathbf{B}_V$ to mark the preserved channels. We further conduct per-token quantization on the sparse residuals to reduce bit-width. Finally, the pruned and quantized residuals $\mathbf{R}_K, \mathbf{R}_V$, along with their binary masks $\mathbf{B}_K, \mathbf{B}_V$ and cluster IDs $\mathbf{I}_K, \mathbf{I}_V$, are stored in off-chip GPU memory. In contrast, the cluster centroids $\mathbf{C}_K$ and $\mathbf{C}_V$ are cached in on-chip shared memory for fast access. Thanks to the high compressibility of centroid-guided residuals extracted from original pre-RoPE KV features, ResKV significantly reduces the KV cache footprint and off-chip memory traffic during inference.

4.2 ResKV-based Flash Decoding Inference

Algorithm 1 details the plug-and-play integration of ResKV with standard Flash Decoding Dao et al. [2023]. The core insight is to perform *differential residual processing* for Key and Value vectors, which minimizes computation while maintaining full compatibility with chunk-wise parallel decod-

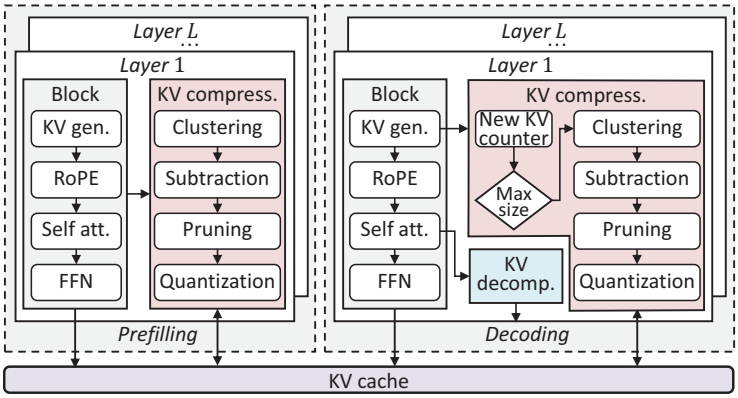


Figure 4: The overall inference pipeline integrated with ResKV.

ing. During inference, sparse residual vectors $\mathbf{R}_K, \mathbf{R}_V$ and bitmaps $\mathbf{B}_K, \mathbf{B}_V$ are decompressed on-chip without modifying the original Flash Decoding architecture.

For Key vectors, since the RoPE operation requires complete dense Key vectors, we must restore full Key vectors by adding residuals to their centroids $\mathbf{C}_K$ before computing attention scores. In contrast, Value vectors are processed far more efficiently: only the residual $\mathbf{R}_V$ is restored, and centroid $\mathbf{C}_V$ is *not* added back immediately. Instead, weighted cluster weights are accumulated in parallel across chunks and only fused globally after cross-chunk reduction.

This deferred centroid fusion drastically reduces per-token computation. After accumulating weighted residuals and cluster statistics, all chunk states are merged. The Value centroids are applied only once in a global batch operation, avoiding heavy per-token computation. By decoupling Key and Value processing, ResKV maintains high accuracy while reducing computational overhead and memory traffic. This design enables ResKV to act as a lightweight, drop-in enhancement for Flash Decoding.

4.3 Overall Pipeline Integrated with ResKV

As shown in Fig. 4, ResKV applies residual-based compression in both the prefill phase and decoding phase. As we find that K vectors exhibit higher cosine similarity prior to the application of RoPE, we immediately launch residual-based compression after KV generation. In this case, the clustering can not only generate more compact clusters but also be overlapped with other operations, including self-attention, RoPE, and FFN. Moreover, our residual-based KV compression is conducted in a layer-by-layer fashion. In the prefill phase, once KV vectors for all input tokens for one layer are generated, they are immediately compressed and stored back to GPU memory, reducing memory usage and enabling long-context inference. Guided by prior observations regarding attention sinks, we specifically keep the initial tokens uncompressed during this stage to preserve accuracy. In the decoding phase, new KVs are generated in autoregressive mode, and each decoding step will perform KV decompression to restore dense KV vectors. We maintain a dense recent window to facilitate the dynamic compression of these newly generated tokens. When the number of newly generated KVs reaches a predefined threshold, we perform residual-based compression for these new KVs, incorporating existing cluster centroids.

5 Evaluation

5.1 Experimental Setup

Models and Benchmarks. To comprehensively evaluate the accuracy and efficiency of ResKV, we conduct experiments across multiple large language models, including Llama-3-8B, Llama-3.1-8B Dubey et al. [2024], and Qwen3-8B Yang et al. [2025]. We employ LongBench Bai et al. [2024], a benchmark covering diverse real-world scenarios such as Single-Document QA, Multi-Document QA, Summarization, Code Completion, and Synthetic Tasks, to holistically measure compression robustness in varying semantic contexts. Furthermore, we utilize RULER Hsieh et al. [2024], spanning

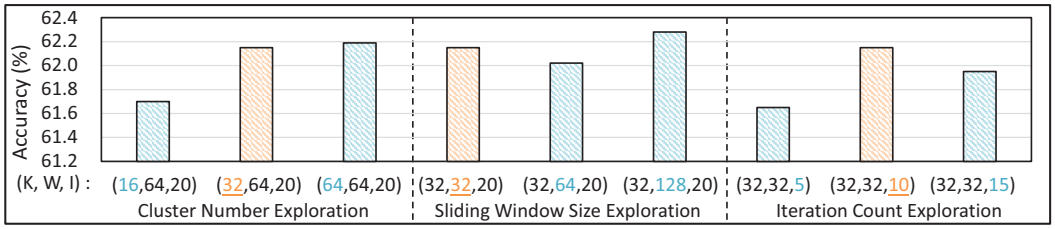


Figure 5: Hyper-parameter exploration of cluster number (K), sliding window size (W), and clustering iteration count (I) on Llama-3.1-8B-Instruct. The evaluation is conducted with a 16K input length on six LongBench tasks: Qasper, 2WikiMQA, QMSum, TriviaQA, Retrieval, and LCC. The reported accuracy is averaged over these tasks.

13 distinct tasks with a 16K input length. Additionally, we incorporate the Needle-In-A-Haystack (NIAH) evaluation to visualize retrieval fidelity via accuracy heatmaps, examining potential information loss across varying document depths.

Baselines. To ensure fair experimental comparison, we evaluate ResKV against two state-of-the-art channel-wise sparsity methods, SparK Liao et al. [2025] and Mustafar Joo et al. [2025], in terms of predictive accuracy. For end-to-end throughput and latency evaluation, we only compare ResKV with Mustafar and the FlashAttention-2-accelerated dense baseline Dao [2024], since the official system implementation of SparK is not publicly available. To guarantee a consistent accuracy baseline across all compared methods, we maintain an identical sliding window size for storing the most recent dense KV entries.

Implementation Details. We evaluate all approaches under identical sparsity levels; specifically, we standardize the configuration by maintaining the initial four tokens as uncompressed sink tokens for all methods to guarantee a fair comparison. All efficiency evaluations are performed on an NVIDIA RTX 6000 GPU using CUDA 13.0.

5.2 Hyper-Parameter Exploration

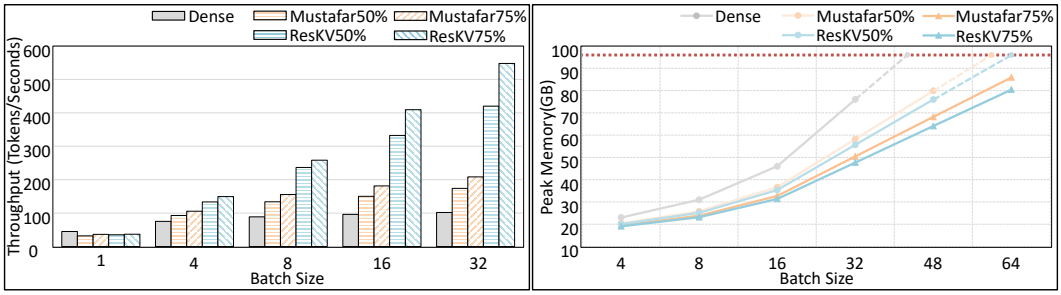
We analyze the sensitivity of ResKV to cluster count K, sliding window size W, and clustering iteration I under the evaluation setting described in Fig. 5. As shown in Fig. 5, increasing K from 16 to 32 boosts the average accuracy from 61.70 to 62.15, while further increasing K to 64 brings only marginal gains (62.19), showing diminishing returns. Thus, $K = 32$ achieves the best trade-off between compression quality and memory cost. For window size, $W = 128$ yields the highest accuracy (62.28), but $W = 32$ maintains strong accuracy with lower computation, making it more efficient. Setting $I = 10$ delivers the best overall accuracy (62.15) compared to 5 and 15 iterations, showing that 10 iterations suffice for feature convergence. Based on this hyper-parameter study, we set $(K = 32, W = 32, I = 10)$ as the default ResKV configuration and keep it fixed in the remaining experiments unless otherwise specified.

5.3 Accuracy Evaluation

Results on LongBench. Table 1 presents the performance of ResKV on Llama-3-8B and Qwen3-8B across different sparsity levels. At 50% sparsity, ResKV achieves results comparable to the dense baseline, retaining 99.6% and 99.8% of the dense baseline accuracy for Llama-3 and Qwen3, respectively. As sparsity increases to 75%, ResKV maintains an average score of 42.46 on Llama-3, outperforming SparK and Mustafar by 2.05 and 2.26 points, respectively. At 87.5% sparsity, ResKV significantly outperforms the baselines. On Llama-3, it achieves an average score of 40.33, exceeding SparK and Mustafar by +6.5 and +7.5 points, respectively. Similarly, on Qwen3, ResKV outperforms SparK by 9.2 points (42.13 vs. 32.89), retaining 93% of the dense baseline performance. These gains arise because pruning low-magnitude residuals preserves semantics far better than standard input/output-based pruning. Accuracy results on RULER and NIAH benchmarks are detailed in Appendix C.

Table 1: Evaluation results of Llama-3-8B-Instruct and Qwen3-8B on the LongBench benchmark with an input length of 8K. We provide a comparison of ResKV with SparK and Mustafar across varying sparsity levels, highlighting the average performance of ResKV.

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	
Llama-3-8B-Instruct																	
Dense Baseline	23.78	43.44	43.19	46.39	38.70	23.26	29.96	22.54	27.76	74.50	90.20	42.16	5.00	70.00	56.87	53.83	43.22
Mustafar (50%)	23.54	43.43	43.14	46.28	39.27	22.85	28.00	22.99	26.82	74.00	90.58	40.15	6.00	69.00	55.48	53.86	42.84
SparK (50%)	23.89	42.75	42.70	46.18	38.95	23.22	29.16	22.60	27.07	74.00	89.81	40.86	5.00	68.00	54.95	49.75	42.43
ResKV (50%)	23.56	43.86	42.33	45.14	39.47	22.03	29.40	22.47	27.64	74.50	90.18	42.03	5.00	69.00	58.20	53.63	**43.03**
Mustafar (75%)	22.95	38.17	40.47	45.51	38.78	22.17	20.59	21.47	22.34	69.00	89.70	37.86	6.00	67.50	50.57	50.13	40.20
SparK (75%)	23.10	39.96	44.59	44.82	37.94	22.63	24.20	21.61	25.76	73.50	89.57	38.05	5.00	68.00	46.60	41.19	40.41
ResKV (75%)	24.54	41.77	42.98	45.43	38.52	22.46	26.43	22.19	26.63	74.50	90.02	40.34	5.00	67.00	57.16	54.34	**42.46**
Mustafar (87.5%)	19.71	29.21	30.86	40.41	31.45	16.50	12.35	20.17	12.47	39.00	80.59	35.56	8.00	64.00	43.91	40.70	32.81
SparK (87.5%)	21.25	30.09	36.57	43.85	32.61	21.31	18.90	20.00	21.76	50.00	89.44	32.50	2.85	62.50	29.25	27.98	33.80
ResKV (87.5%)	23.02	36.89	38.91	44.83	37.84	20.89	22.38	21.44	24.02	67.00	89.62	38.49	5.50	66.50	54.88	53.03	**40.33**
Qwen3-8B																	
Dense Baseline	23.57	47.64	54.58	53.84	44.52	29.60	33.27	23.00	24.72	71.00	89.32	44.45	2.00	74.00	56.09	53.28	45.31
Mustafar (50%)	24.24	46.96	54.53	53.27	44.40	29.96	32.67	22.88	24.71	71.00	89.32	43.16	2.50	74.00	56.33	54.32	45.27
SparK (50%)	23.09	46.88	54.78	52.61	44.81	29.41	32.90	22.85	24.57	71.00	89.32	43.54	1.50	73.50	55.95	53.69	45.03
ResKV (50%)	23.58	47.11	54.61	53.26	44.47	29.80	33.02	22.64	24.71	71.00	89.32	43.55	2.50	74.00	56.18	53.81	**45.22**
Mustafar (75%)	25.05	45.64	52.26	52.31	43.68	29.55	28.54	22.04	21.95	71.00	87.74	40.94	3.50	75.00	54.83	52.34	44.15
SparK (75%)	24.35	45.07	53.06	52.07	44.42	29.08	30.51	21.69	23.25	71.00	89.57	41.60	1.25	75.00	52.33	48.92	43.95
ResKV (75%)	24.07	44.65	52.78	52.65	44.06	29.65	31.39	22.70	24.29	71.00	89.82	42.28	1.00	73.50	55.04	53.21	**44.51**
Mustafar (87.5%)	20.77	32.64	40.30	47.26	39.14	22.61	18.80	19.98	14.84	42.00	85.13	35.48	2.50	71.00	48.51	46.09	36.69
SparK (87.5%)	16.68	29.46	36.90	36.31	26.04	18.11	23.31	18.74	18.22	57.50	87.24	31.77	0.00	55.68	35.58	34.67	32.89
ResKV (87.5%)	21.73	42.80	46.83	51.55	42.12	26.16	26.94	21.99	21.96	66.50	87.26	40.89	1.67	72.00	53.87	49.83	**42.13**



(a) Generation throughput.

(b) Peak memory usage.

Figure 6: Efficiency evaluation of ResKV on Llama-3.1-8B-Instruct across varying batch sizes. Experiments are configured with an input length of 8192 and an output length of 2048.

5.4 Efficiency Evaluation

We evaluate the system-level efficiency of ResKV by analyzing its impact on two critical dimensions: end-to-end throughput and memory scalability.

Generation Throughput. We evaluate the end-to-end generation throughput configured with 8192 input and 2048 output tokens. Benefiting from our aggressive memory footprint reduction and fused kernel optimizations, ResKV achieves remarkable generation throughput. At a batch size of 16, ResKV (75% sparsity) secures a massive 4.27× speedup over the dense baseline. As the batch size scales to 32, ResKV at 75% sparsity achieves a 2.63× end-to-end throughput speedup over Mustafar at the same sparsity. Ultimately, this demonstrates that our method effectively translates theoretical memory savings into substantial end-to-end acceleration in high-concurrency scenarios.

Memory Footprint and Scalability. Fig. 6(b) reports peak memory under the same 8192/2048-token setting, including runtime KV-cache-related buffers such as sparse residuals, binary masks, cluster IDs, quantization scales, centroids, and dense sink/recent-window KV states. At batch size 32, ResKV with 75% sparsity reduces peak memory from 75.97 GB for the dense baseline to 47.70 GB, a 37.2% reduction. It further scales to batch size 48 with 64.07 GB, lower than Mustafar's 68.17 GB at the same sparsity.

Time-To-First-Token (TTFT) and Time-Per-Output-Token (TPOT). We further analyze TTFT and TPOT in Appendix B.1. The results show that ResKV introduces an acceptable one-time prefill overhead from clustering, while substantially reducing decoding latency in the memory-bound generation phase.

Speedup Breakdown. We provide a detailed system performance breakdown in Appendix B.2. The results show that fusing decompression into Flash Decoding and applying centroid-related optimization consistently improve throughput, especially at larger batch sizes. A detailed breakdown of KV-cache storage and compression overhead is provided in Appendix E

5.5 Orthogonality with Other Compression Strategies

We evaluate the orthogonality of ResKV with low-bit quantization and token-wise sparsity to verify its compatibility with other KV cache compression paradigms. Results show that ResKV maintains robust accuracy under joint compression settings, benefiting from its residual-based pruning mechanism. We provide the detailed experimental results and analysis in Appendix D.

6 Related Work

Token-wise Sparsity. This direction aims to reduce the number of tokens involved in memory access. Eviction-based strategies Zhang et al. [2023], Xiao et al. [2024] filter critical tokens via sliding windows or accumulated attention scores. While demonstrating superior performance on long sequences, their permanent discarding mechanism inevitably leads to the irreversible loss of long-tail information. SnapKV Li et al. [2024] compresses the KV cache by selecting important token positions, while Quest Tang et al. [2024] selects query-relevant KV pages for attention. Recallable compression methods such as ClusterKV Liu et al. [2025] and PQCache Zhang et al. [2025a] combine KV selection with CPU offloading and GPU caching. When CPU–GPU transfers are required, they introduce additional data-movement overhead.

Quantization. Quantization reduces memory footprint by compressing the data bit-width. To address significant outliers in Key matrices, existing methods predominantly employ mixed-precision schemes: applying Per-Channel quantization to Key to accommodate outliers, and Per-Token quantization to Value Liu et al. [2024a], Hooper et al. [2024], He et al. [2024]. Despite achieving low-bit compression, these methods face complex dequantization overheads and cumulative precision errors that are difficult to eliminate in ultra-long context scenarios. Crucially, these dense-optimized quantization schemes are inherently incompatible with unstructured channel pruning, as the induced dynamic sparsity patterns disrupt the contiguous memory layout and cause the retrieval of precision-preserving scales to become a hardware bottleneck.

Channel-wise Pruning. This paradigm exploits feature redundancy to compress the KV cache. Structured pruning methods Zhang et al. [2025b], Xu et al. [2025] enforce the removal of identical channels across grouped tokens to ensure hardware efficiency; however, such strict constraints limit the upper bound of lossless sparsity. While unstructured pruning breaks this ceiling, current works Liao et al. [2025], Joo et al. [2025] face dual limitations: (1) They fail to leverage the intrinsic similarity between Key vectors, leading to reconstruction errors under extreme sparsity; (2) They suffer from poor orthogonality with quantization. Per-token quantization incurs high precision loss, while per-channel quantization impedes hardware efficiency. Driven by dynamic sparsity patterns, the latter results in either non-contiguous memory accesses or load imbalance, making effective acceleration intractable.

7 Conclusion and Discussion

We present ResKV, a residual-based channel-wise pruning method designed to tackle the KV cache bottleneck in long-context and big-batch LLM inference. By clustering semantically similar KV vectors and computing low-magnitude centroid residual vectors, ResKV effectively incorporates vector similarity into the pruning pipeline. Performing channel-wise unstructured pruning and per-token quantization on these residual vectors introduces only negligible approximation errors. Extensive experiments confirm that ResKV significantly reduces peak memory consumption, improves inference throughput, and delivers higher model accuracy and output quality than state-of-the-art

channel-wise KV pruning methods under the same sparsity constraints. Beyond technical merits, ResKV contributes to green AI by lowering GPU memory usage, while also democratizing access to long-context LLMs for broader research and real-world applications. Despite these advantages, ResKV also has two limitations: clustering brings mild extra latency in the prefill stage, and when the number of clusters reaches the predefined maximum, newly generated KV tokens that are dissimilar to existing centroids may reduce the accuracy gain of residual-based pruning.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Natural Science Foundation of China under Grant 62274019, and by the Chongqing Higher Education Teaching Reform Research Project (General) under Grant 263006.

References

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling Transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 5, pages 606–624, 2023.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 34661–34710, 2023.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning (ICML)*, 2024.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 22947–22970, 2024.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025a.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning (ICML)*, volume 235, pages 32332–32344, 2024a.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. ZipCache: Accurate and efficient KV cache quantization with salient token identification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 68287–68307, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 1270–1303, 2024.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. ThinK: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations (ICLR)*, 2025.

Donghyeon Joo, Helya Hosseini, Ramyad Hadidi, and Bahar Asgari. MUSTAFAR: Promoting unstructured sparsity for KV cache pruning in LLM inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yike Zhang, Zhiyuan He, Huiqiang Jiang, Chengruidong Zhang, Yuqing Yang, Jianyong Wang, and Lili Qiu. LeanK: Learnable K cache channel pruning for efficient decoding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 31122–31137, 2025b.

Huanxuan Liao, Yixing Xu, Shizhu He, Guanchen Li, Xuanwu Yin, Dong Li, Emad Barsoum, Jun Zhao, and Kang Liu. SparK: Query-aware unstructured sparsity with recoverable KV cache channel pruning. *arXiv preprint arXiv:2508.15212*, 2025.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–7, 2025.

Jie Hu, Shengnan Wang, Yutong He, Ping Gong, Jiawei Yi, Juncheng Zhang, Youhui Bai, Renhai Chen, Gong Zhang, Cheng Li, and Kun Yuan. Efficient long-context LLM inference via KV cache clustering. *arXiv preprint arXiv:2506.11418*, 2025.

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. In *International Conference on Machine Learning (ICML)*, volume 267, pages 57355–57373, 2025.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. PyTorch Blog, October 2023. URL `https://pytorch.org/blog/flash-decoding/`.

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

Ruikang Liu, Haoli Bai, Haokun Lin, Yuening Li, Han Gao, Zhengzhuo Xu, Lu Hou, Jun Yao, and Chun Yuan. IntactKV: Improving large language model quantization by keeping pivot tokens intact. In *Findings of the Association for Computational Linguistics (ACL)*, pages 7716–7741, 2024b.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3119–3137, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024.

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

Simla Burcu Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi, Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, and Amir Yazdanbakhsh. Effective interplay between sparsity and quantization: From theory to practice. In *International Conference on Learning Representations (ICLR)*, 2025.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025. URL `https://openreview.net/forum?id=ayi7qezU87`.

A Visualization Analysis for the Value Cache

Despite value cache exhibiting less evident local similarity compared to key cache, we explore its global redundancy using the same clustering strategy. The results validate this approach: ResKV achieves a low average cosine distance of 0.05. Furthermore, the residual calculation reduces the average magnitude range from 1.38 to 0.6, confirming that exploiting global similarity is equally effective for compressing and quantizing the value cache. Since Mustafar and SparK adopt the same pruning strategy for the Value cache, their corresponding curves overlap in the figure.

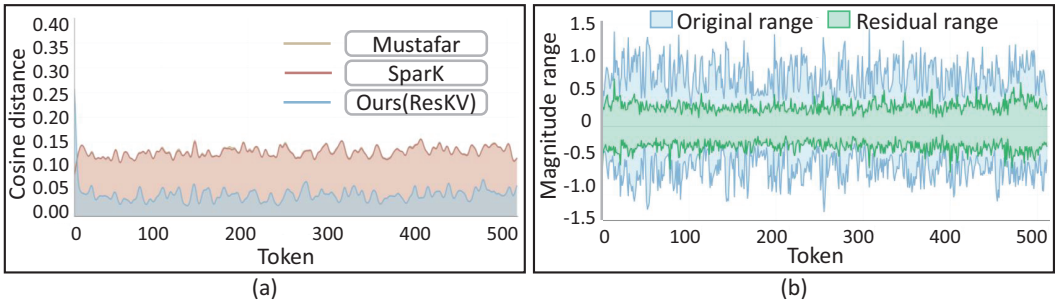


Figure 7: Motivation analysis for the Value cache on Llama-3-8B-Instruct using WikiText-2 (Layer 16, Head 1). (a) The cosine distance between the original Value vector and the pruned Value vector of different approaches at 75% sparsity. (b) The feature magnitude of the original Value vector and the residual Value vector.

B Detailed Efficiency Breakdown

In this section, we provide detailed efficiency analysis of ResKV, including the latency breakdown of TTFT and TPOT, as well as the system performance breakdown of different implementation components. These results complement the main-text efficiency evaluation by showing how the proposed system design affects both prefill and decoding latency.

B.1 TTFT and TPOT

Time-to-First-Token (TTFT) reflects the one-time cost of processing the input context during prefill, while Time-Per-Output-Token (TPOT) measures the per-token latency during autoregressive decoding.

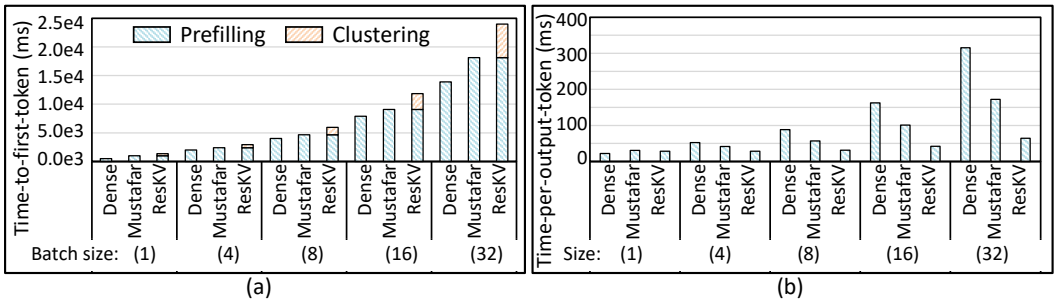


Figure 8: Latency breakdown across methods with 50% sparsity under the same system setting as Fig. 6: Llama-3.1-8B-Instruct on an NVIDIA RTX 6000 GPU with CUDA 13.0, configured with an input length of 8192 and an output length of 2048. (a) Comparison of Time-to-First-Token (TTFT), and (b) comparison of Time-Per-Output-Token (TPOT).

As shown in Fig. 8(a), ResKV introduces additional TTFT due to the clustering stage in the prefill phase. By limiting clustering to 10 iterations, this overhead remains controlled: across the evaluated

batch sizes and sparsity settings, clustering accounts for 19.15% to 27.84% of the total TTFT. This indicates that the global feature aggregation introduces an acceptable one-time prefill cost.

The additional prefill cost is compensated by faster decoding. As shown in Fig. 8(b), ResKV reduces TPOT in the memory-bound generation phase. At batch size 32 and 50% sparsity, ResKV achieves a TPOT of 64.34 ms, compared with 172.13 ms for Mustafar at the same sparsity, corresponding to a 2.68× decoding speedup. This TPOT speedup measures decoding only, whereas the 2.63× end-to-end throughput speedup in Section 5.4 is measured at 75% sparsity and includes prefill. This improvement mainly comes from two designs: ResKV fuses decompression into the Flash Decoding kernel to avoid redundant intermediate memory accesses, and it applies deferred centroid fusion for Value vectors so that centroid compensation is performed once after global reduction rather than for every token.

B.2 System Improvement Breakdown

We further provide a system speedup breakdown to quantify the contribution of different implementation components. In our unfused ResKV baseline, sparse KV states are first restored in GPU memory and then consumed by the attention kernel. This introduces additional memory read/write traffic for intermediate dense states, which becomes costly in memory-bound decoding.

To reduce this overhead, ResKV fuses residual decompression into the Flash Decoding kernel, denoted as *+FlashDecoding*. This design avoids materializing intermediate dense KV states in global memory. We further introduce centroid-related optimization for the Value cache, denoted as *+ValueDeferredFusion*, which avoids repeated centroid compensation inside the per-token decoding loop by applying centroid fusion after global reduction.

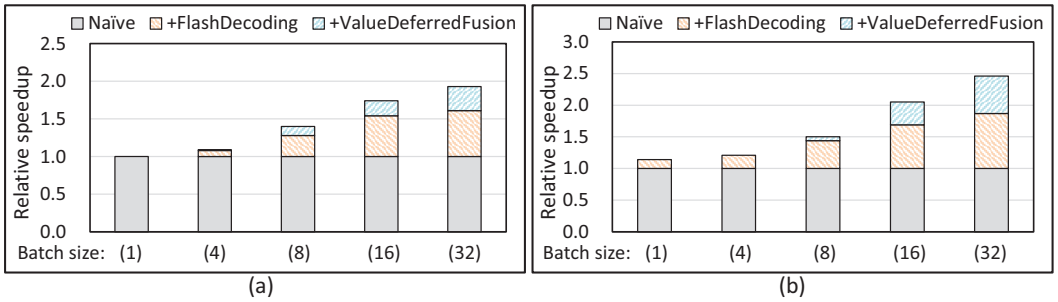


Figure 9: Relative speedup breakdown of ResKV under the same system setting as Fig. 6: Llama-3.1-8B-Instruct on an NVIDIA RTX 6000 GPU with CUDA 13.0, configured with an input length of 8192 and an output length of 2048. (a) Results at 50% sparsity, and (b) results at 75% sparsity.

As shown in Fig. 9, *+FlashDecoding* consistently improves throughput over the naive implementation by eliminating unnecessary intermediate memory accesses. When combined with *+ValueDeferredFusion*, ResKV achieves the highest speedup, with the improvement becoming more evident at larger batch sizes such as $B = 16$ and $B = 32$. This trend is consistent with the memory-bound nature of decoding: as batch size increases, reducing off-chip memory traffic and avoiding repeated centroid operations become increasingly important for improving end-to-end throughput.

Overall, the performance breakdown shows that the efficiency gain of ResKV mainly comes from reducing memory movement and fusing decompression with attention computation, rather than relying on a separate decompression stage before dense attention.

C Accuracy Evaluation on Additional Benchmarks

In this section, we provide supplementary accuracy evaluations on RULER and Needle-In-A-Haystack (NIAH). These benchmarks complement the LongBench results in the main text by evaluating ResKV under long-context retrieval and synthetic reasoning scenarios.

C.1 Results on RULER

Table 2: Evaluation results of Llama-3.1-8B-Instruct and Qwen3-8B on the RULER benchmark. We provide a comparison of ResKV with SparK and Mustafar at an input length of 16K across varying sparsity levels. We highlight the average performance of ResKV.

Method	Niah1	Niah2	Niah3	MKey1	MKey2	MKey3	MValue	MQuery	VT	CWE	FWE	QA1	QA2	Avg
Llama-3.1-8B-Instruct														
Dense (Full)	100.0	100.0	100.0	100.0	100.0	100.0	99.25	99.25	99.2	80.7	90.33	87.0	55.0	93.13
Sparsity 50%														
Mustafar	100.0	100.0	100.0	100.0	100.0	100.0	99.50	99.75	99.2	70.5	90.00	87.0	55.0	92.38
SparK	100.0	100.0	100.0	100.0	100.0	100.0	99.25	99.50	99.2	78.9	90.67	87.0	56.0	93.12
ResKV	100.0	100.0	100.0	100.0	100.0	100.0	99.50	99.25	99.0	80.0	90.67	87.0	55.0	**93.11**
Sparsity 75%														
Mustafar	90.0	92.0	67.0	93.0	96.0	39.0	88.75	93.75	68.0	6.9	86.33	84.0	50.0	73.44
SparK	100.0	97.0	92.0	94.0	84.0	61.0	97.25	95.00	85.0	4.5	90.33	83.0	52.0	79.62
ResKV	100.0	100.0	100.0	99.0	100.0	99.0	97.25	98.75	95.6	50.9	91.33	86.0	54.0	**90.14**
Sparsity 87.5%														
Mustafar	0.0	8.0	2.0	11.0	3.0	0.0	4.25	2.00	2.8	0.0	72.00	59.0	37.0	15.47
SparK	0.0	3.0	0.0	0.0	0.0	0.0	2.75	0.75	0.0	0.2	50.00	25.0	25.0	8.21
ResKV	3.0	9.0	3.0	28.0	94.0	31.0	6.25	9.75	13.8	4.2	92.33	79.0	48.0	**32.41**
Qwen3-8B														
Dense (Full)	100.0	100.0	100.0	100.0	100.0	100.0	97.50	98.75	98.2	90.8	91.67	77.0	51.0	92.69
Sparsity 50%														
Mustafar	100.0	100.0	100.0	100.0	100.0	100.0	98.50	99.00	98.2	89.2	91.33	76.0	52.0	92.63
SparK	100.0	100.0	100.0	100.0	100.0	100.0	97.00	98.00	95.6	87.9	91.67	79.0	51.0	92.32
ResKV	100.0	100.0	100.0	100.0	100.0	100.0	98.00	99.25	96.0	89.2	93.67	76.0	52.0	**92.62**
Sparsity 75%														
Mustafar	100.0	100.0	99.0	100.0	100.0	95.0	98.75	95.50	66.6	69.7	93.67	74.0	48.0	87.71
SparK	100.0	100.0	99.0	100.0	100.0	100.0	94.00	96.50	84.6	57.8	93.33	78.0	53.0	88.94
ResKV	100.0	100.0	100.0	100.0	100.0	99.0	99.00	98.50	94.8	77.1	94.33	75.0	51.0	**91.44**
Sparsity 87.5%														
Mustafar	71.0	34.0	35.0	44.0	24.0	3.0	26.00	7.25	3.4	3.3	67.33	53.0	38.0	31.48
SparK	100.0	98.0	92.0	89.0	47.0	21.0	88.75	86.75	75.0	20.7	85.33	56.0	45.0	69.58
ResKV	98.0	90.0	88.0	95.0	100.0	6.0	86.50	80.75	48.4	35.6	93.67	64.0	43.0	**71.46**

Table 2 reports the results on RULER using Llama-3.1-8B-Instruct and Qwen3-8B with a 16K input length. At 50% sparsity, ResKV closely matches the dense baseline on both models. Specifically, ResKV achieves an average score of 93.11 on Llama-3.1-8B-Instruct, compared with 93.13 for the dense baseline. On Qwen3-8B, ResKV obtains 92.62, also close to the dense baseline score of 92.69. These results indicate that residual-based channel-wise pruning introduces negligible accuracy loss at moderate sparsity.

As the sparsity level increases, ResKV shows stronger robustness than existing channel-wise pruning baselines. At 75% sparsity on Llama-3.1-8B-Instruct, ResKV achieves an average score of 90.14, outperforming Mustafar and SparK by 16.70 and 10.52 points, respectively. On Qwen3-8B, ResKV achieves 91.44, also higher than Mustafar and SparK. The advantage is particularly clear on complexity-sensitive tasks. For example, on the CWE task with Llama-3.1-8B-Instruct at 75% sparsity, Mustafar and SparK drop to 6.9 and 4.5, respectively, while ResKV maintains a score of 50.9.

At the extreme 87.5% sparsity level, all methods experience degradation, but ResKV still preserves more useful information than the baselines. On Llama-3.1-8B-Instruct, ResKV achieves an average score of 32.41, outperforming Mustafar and SparK by 16.94 and 24.20 points, respectively. On Qwen3-8B, ResKV reaches an average score of 71.46, exceeding Mustafar by 39.98 points and remaining higher than SparK. These results further demonstrate that pruning low-magnitude centroid residuals provides a more stable sparsity-accuracy trade-off under aggressive compression.

C.2 Results on Needle-In-A-Haystack

We further evaluate Meta-Llama-3.1-8B-Instruct on the Needle-In-A-Haystack benchmark across context lengths from 1K to 24K. The evaluation includes six compression configurations: 50% sparsity, 75% sparsity, 50% sparsity with INT4 quantization, 75% sparsity with INT4 quantization, 50% sparsity with INT2 quantization, and 75% sparsity with INT2 quantization.

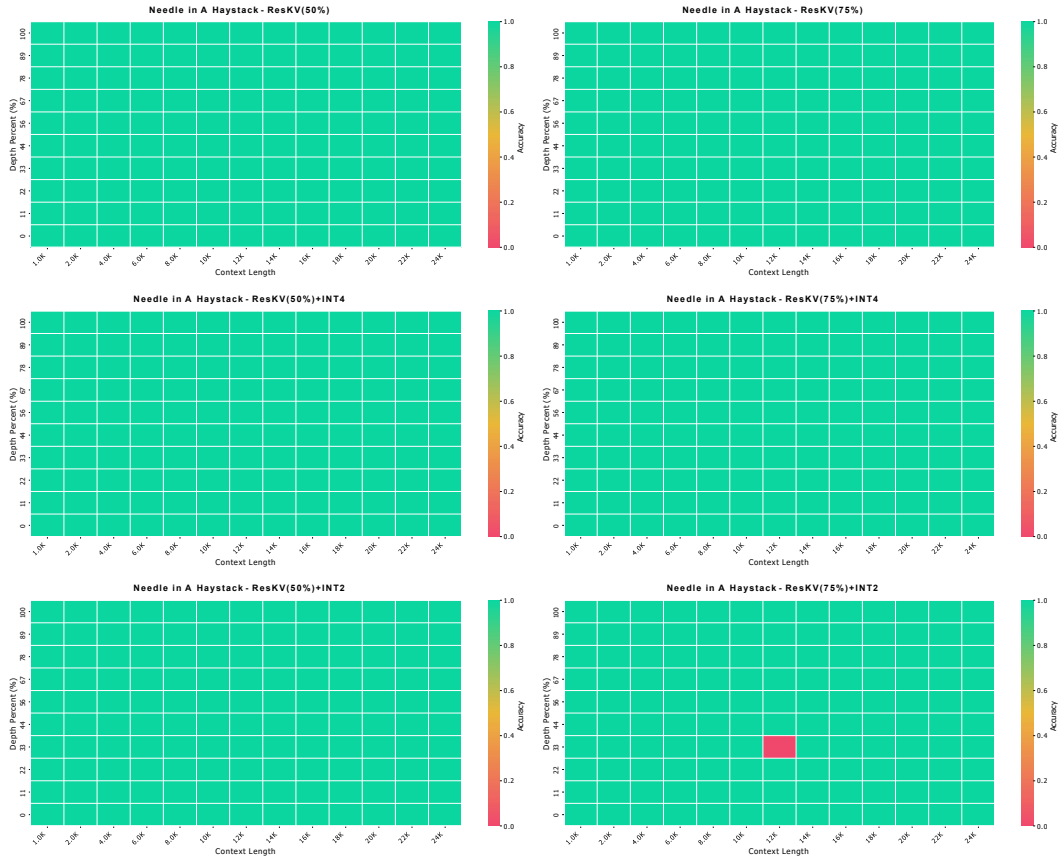


Figure 10: **Needle-In-A-Haystack results.** We evaluate ResKV under different sparsity and quantization configurations across context lengths from 1K to 24K. Green indicates 100% accuracy, while red indicates failure.

As shown in Fig. 10, ResKV maintains stable retrieval performance across almost all evaluated settings. The sparse-only configurations and the INT4 joint-compression configurations achieve 100% accuracy across all tested context lengths and needle depths. With INT2 quantization, ResKV also maintains robust retrieval accuracy at 50% sparsity.

The only observed failure occurs under the most aggressive configuration, 75% sparsity combined with INT2 quantization, at a context length of 12K and a depth of 33%. This indicates that ResKV largely preserves long-context retrieval capability under both sparsity and quantization, while occasional retrieval errors may appear when aggressive channel pruning is combined with very low-bit quantization. Overall, the NIAH results support the robustness of ResKV in long-context retrieval scenarios.

D Orthogonality with Other Compression Strategies

In this section, we provide supplementary accuracy results to evaluate the orthogonality of ResKV with other KV cache compression strategies, including low-bit quantization and token-wise sparsity. These experiments examine whether residual-based channel-wise pruning can be combined with additional compression methods while preserving model accuracy.

D.1 Joint Application with Quantization

We first evaluate the compatibility between ResKV and low-bit quantization. While per-channel quantization is commonly adopted in dense KV cache compression pipelines, it is less compatible with channel-wise unstructured sparsity because different tokens may preserve different channel subsets. Therefore, we adopt per-token quantization, which naturally aligns with token-dependent

sparsity patterns. Following the findings of Harma et al. [2025], quantization is applied after sparsity. Since ResKV operates on low-magnitude centroid residuals rather than raw KV features, the retained values are less affected by activation outliers, which helps reduce quantization error under low-bit settings.

Table 3: Evaluation results of Llama-3-8B-Instruct and Qwen3-8B on the LongBench benchmark with an input length of 8K. Experiments are configured to compare different methods at 50% sparsity across varying quantization bit-widths. We highlight the average performance of ResKV.

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	
Dense Baseline	23.78	43.44	43.19	46.39	38.70	23.26	29.96	22.54	27.76	74.50	90.20	42.16	5.00	70.00	56.87	53.83	43.22
Llama-3-8B-Instruct																	
Sparsity 50% + INT4																	
Mustafar	23.39	42.65	43.01	46.35	38.81	23.26	28.58	22.83	27.00	74.00	89.95	40.95	6.00	68.00	54.80	52.71	42.64
SparK	23.89	42.80	42.58	46.18	38.95	23.22	29.94	22.22	27.26	74.00	89.95	41.42	5.50	67.50	54.54	50.95	42.56
ResKV	23.56	43.13	42.54	45.87	39.65	23.43	29.21	22.90	27.42	74.50	90.19	41.27	4.50	69.50	57.38	53.79	**43.05**
Sparsity 50% + INT2																	
Mustafar	21.99	34.11	34.57	42.44	34.42	17.67	15.58	20.91	21.66	69.00	81.59	37.89	8.00	60.50	34.62	30.84	35.36
SparK	21.04	32.32	35.81	39.71	34.42	20.07	17.41	17.45	24.08	63.00	75.11	16.19	7.56	53.00	35.44	30.02	32.66
ResKV	24.32	41.53	41.41	44.67	38.24	23.09	29.62	22.74	27.69	74.50	89.96	41.19	6.50	66.50	56.64	53.09	**42.61**
Qwen3-8B																	
Dense Baseline	23.57	47.64	54.58	53.84	44.52	29.60	33.27	23.00	24.72	71.00	89.32	44.45	2.00	74.00	56.09	53.28	45.31
Sparsity 50% + INT4																	
Mustafar	24.17	46.64	54.52	53.31	44.52	30.03	32.98	22.51	24.53	71.00	89.41	42.61	2.00	74.50	56.23	53.92	45.18
SparK	22.07	46.78	53.97	53.09	44.85	29.64	32.27	22.42	24.49	71.00	89.32	43.03	1.58	74.00	54.38	51.46	44.65
ResKV	23.41	47.11	54.13	53.60	44.58	29.86	32.73	22.89	24.67	71.00	89.32	44.11	1.50	74.00	55.55	53.63	**45.13**
Sparsity 50% + INT2																	
Mustafar	22.49	42.09	49.11	52.56	42.58	28.18	29.88	21.88	23.31	70.00	87.92	40.99	2.00	63.00	52.60	48.71	42.33
SparK	12.45	24.13	33.09	21.96	18.03	10.45	25.53	22.23	22.53	63.75	55.91	12.79	0.50	41.92	19.19	21.04	25.35
ResKV	23.97	46.05	51.57	52.48	44.20	30.28	31.26	23.04	24.05	71.00	89.82	42.68	0.00	66.50	55.02	52.50	**44.03**

As shown in Table 3, when 50% sparsity is combined with INT4 quantization, ResKV achieves accuracy comparable to its sparse-only counterpart. On Llama-3-8B-Instruct, ResKV obtains an average score of 43.05 under 50% sparsity plus INT4, close to the sparse-only result of 43.03. On Qwen3-8B, ResKV achieves 45.13, also close to the sparse-only result of 45.22. These results indicate that INT4 quantization introduces negligible additional accuracy loss when applied to residual-based sparse KV representations.

Under the more aggressive INT2 setting, the advantage of residual-based compression becomes more evident. ResKV achieves average scores of 42.61 and 44.03 on Llama-3-8B-Instruct and Qwen3-8B, respectively, retaining 98.6% and 97.2% of the corresponding dense baseline accuracy. In contrast, the baselines suffer larger degradation, especially on Qwen3-8B, where SparK drops to an average score of 25.35. These results suggest that quantizing low-magnitude residual offsets is more robust than quantizing raw KV features under very low bit-widths.

D.2 Joint Application with Token-wise Sparsity

We next evaluate the compatibility between ResKV and token-wise sparsity. Token-wise eviction methods reduce the number of retained KV tokens, while channel-wise pruning compresses the feature dimension of each retained token. These two strategies are complementary in principle. However, directly combining token eviction with conventional channel-wise pruning may distort the feature representations required for accurate token retrieval. ResKV mitigates this issue by pruning low-magnitude residuals around cluster centroids, which better preserves the semantic information of retained tokens.

Table 4: Integration of ResKV with SnapKV and PyramidKV on the LongBench benchmark (Llama-3.1-8B-Instruct, 16K context, KV budget 512). ResKV consistently outperforms SparK and Mustafar at both 50% and 75% sparsity levels.

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	
Dense Baseline	26.21	45.27	53.78	54.76	47.13	30.92	34.56	24.89	27.45	72.50	91.48	43.67	6.33	99.50	63.36	56.53	48.65
SnapKV (KV budget 512)																	
SnapKV	26.28	41.30	51.70	54.38	45.09	29.99	25.49	23.50	24.78	60.00	91.63	41.75	5.92	99.50	62.36	54.97	46.16
+SparK (50%)	25.95	40.52	50.95	54.42	45.35	29.50	24.80	23.25	24.65	59.50	91.80	41.35	5.75	99.50	62.55	54.30	45.88
+Mustafar (50%)	26.10	40.84	50.84	54.47	45.53	29.69	24.79	22.85	24.68	59.00	91.92	40.83	6.08	99.50	62.68	54.95	45.92
+ResKV (50%)	26.06	40.53	50.98	54.60	45.26	30.10	25.01	23.53	24.82	60.00	91.71	41.47	5.92	99.50	62.72	54.54	**46.05**
+SparK (75%)	25.98	38.72	50.68	54.40	45.19	29.68	23.58	22.93	23.85	58.50	92.67	38.49	5.42	99.50	60.29	51.79	45.10
+Mustafar (75%)	24.75	36.39	46.89	54.67	44.09	29.14	22.28	22.60	22.25	49.00	91.22	40.15	6.75	99.00	58.68	51.76	43.73
+ResKV (75%)	25.46	38.39	50.13	54.89	44.61	29.25	23.97	23.30	24.48	58.00	91.70	41.22	5.78	99.50	61.98	54.60	**45.45**
PyramidKV (KV budget 512)																	
PyramidKV	26.22	44.76	52.22	56.22	46.77	29.08	31.32	24.14	27.19	69.50	91.42	44.08	6.17	100.00	62.87	54.02	47.87
+SparK (50%)	25.90	44.50	51.20	55.70	46.20	28.50	30.20	23.70	26.60	68.00	91.00	43.20	6.00	99.50	62.60	53.20	47.25
+Mustafar (50%)	26.00	44.50	51.50	55.80	46.50	28.50	30.50	23.80	26.80	68.00	91.00	43.50	6.00	100.00	62.00	53.50	47.37
+ResKV (50%)	26.17	44.30	50.33	55.79	47.02	28.77	30.75	23.87	27.40	69.00	91.41	43.49	7.00	100.00	63.02	54.45	**47.67**
+SparK (75%)	25.20	40.80	51.00	54.80	45.20	28.80	25.50	23.80	25.00	63.90	90.80	42.00	6.00	99.50	61.20	52.50	46.00
+Mustafar (75%)	24.80	40.00	50.50	54.50	45.00	27.80	24.50	23.00	24.00	61.00	90.00	41.00	5.50	99.00	60.00	51.50	45.13
+ResKV (75%)	25.49	41.53	49.04	54.99	46.32	29.25	27.83	23.67	26.60	67.50	89.90	43.13	6.55	99.50	62.10	54.30	**46.73**

As shown in Table 4, we integrate ResKV with SnapKV and PyramidKV Cai et al. [2025] under a KV budget of 512 tokens. With SnapKV, adding ResKV at 50% channel-wise sparsity achieves an average score of 46.05, which is close to the pure SnapKV baseline of 46.16 and slightly higher than the corresponding SparK and Mustafar combinations. At 75% sparsity, SnapKV combined with ResKV obtains 45.45, outperforming SnapKV combined with SparK and Mustafar, which achieve 45.10 and 43.73, respectively.

A similar trend is observed with PyramidKV. At 50% channel-wise sparsity, PyramidKV combined with ResKV achieves an average score of 47.67, with only a small drop from the pure PyramidKV baseline of 47.87 and higher accuracy than the SparK and Mustafar combinations. At 75% sparsity, ResKV maintains an average score of 46.73, outperforming SparK and Mustafar under the same token-wise sparsity setting. These results demonstrate that ResKV can be jointly applied with token-wise sparsity while maintaining stronger accuracy than existing channel-wise pruning baselines, especially under higher sparsity.

E KV-Cache Storage and Runtime Overhead

KV-cache storage. Table 5 reports cache-only storage at batch size 16 with 8,192 input tokens and 2,048 output tokens, using $K = W = 32$ and four uncompressed sink tokens. Including the storage of centroids, masks, cluster IDs, scales, and the dense window, ResKV at 75% sparsity achieves $3.11\times$ compression with BF16 residuals and $7.00\times$ with INT4 residuals.

Table 5: KV-cache-only storage breakdown on Llama-3.1-8B-Instruct. Storage is reported in GB, excluding model parameters and non-cache runtime allocations. Dense KV denotes the complete cache for Dense BF16 and the sink/recent window for ResKV. The compression ratio is the dense KV-cache size divided by the compressed KV-cache size.

Method	Centroids (GB)	Residuals (GB)	Bit masks (GB)	Cluster IDs (GB)	Quant. scales (GB)	Dense KV (GB)	Total (GB)	Ratio ($\times$)
Dense BF16	–	–	–	–	–	20.0000	20.0000	1.00
ResKV 75%, BF16	0.0625	4.9824	1.2456	0.0779	–	0.0703	6.4387	3.11
ResKV 75%, INT4	0.0625	1.2456	1.2456	0.0779	0.1557	0.0703	2.8576	7.00

Compression overhead. Table 6 reports compression latency at batch size 32 with 8,192 input tokens and 2,048 output tokens. Total compression overhead accounts for 5.08%–5.54% of end-to-end latency; nearest-centroid search and residual compression during generation together account for 0.21%–0.25%.

Table 6: Runtime breakdown on Llama-3.1-8B-Instruct at 75% sparsity. Latencies are in milliseconds; parentheses give the percentage of end-to-end latency. End-to-end latency includes all listed compression operations.

	Prefill		Generation			
Method	Clustering (ms)	Residual compression (ms)	Centroid search (ms)	Residual compression (ms)	Total overhead (ms)	End-to-end latency (ms)
ResKV 75%, BF16	5,945.54 (4.78%)	110.78 (0.09%)	214.80 (0.17%)	47.98 (0.04%)	6,319.10 (5.08%)	124,373.56
ResKV 75%, INT4	5,949.42 (5.17%)	134.05 (0.12%)	228.59 (0.20%)	54.53 (0.05%)	6,366.59 (5.54%)	115,015.41

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state ResKV's core contributions including residual-driven channel-wise pruning, joint optimization of pruning and quantization, and integration with Flash Decoding, which are fully supported by method description and experimental results in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper has limitations including slightly increased prefill latency from clustering and degraded accuracy when dissimilar new KV tokens are assigned to full clusters, and these are discussed in the section of conclusion and discussion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper mainly focuses on empirical evaluation and engineering design without formal theoretical results, theorems, or proofs. Thus, no assumptions or proof completeness need to be checked.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper fully discloses detailed experimental settings, including model architectures, benchmarks, hyperparameters, and implementation configurations, enabling researchers to reproduce the main experimental results. All key parameters and evaluation procedures are clearly stated to support the verification of the core claims and conclusions.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The experiments use publicly available models and established benchmarks, including LongBench, RULER, and Needle-In-A-Haystack. The implementation, installation instructions, and accuracy and performance evaluation procedures are publicly available at `https://github.com/coralabo/ResKV`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper clearly specifies all essential experimental details, including test benchmarks, hyperparameters, cluster settings, and evaluation environments, which are sufficient to understand and interpret the experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: Our experiments focus on fixed hyperparameter settings and deterministic evaluation pipelines with consistent model seeds; error bars or statistical tests are not included.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the GPU type (NVIDIA RTX 6000), CUDA version, and reports peak GPU memory usage across experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper focuses on LLM KV cache compression without involving human subjects, privacy data, bias, harmful applications or ethical violations, fully complying with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses positive societal impacts including green AI and broader accessibility of longcontext LLMs in the section of conclusion and discussion.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work focuses on LLM KV cache compression optimization and does not involve data collection, model training, or release of high-risk artifacts, so no additional safeguards are required.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper properly cites all original works for the models, datasets, and baselines used, and respects their corresponding licenses and terms of use without unauthorized usage or misappropriation.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We release the ResKV implementation under the Apache-2.0 license. The repository provides documentation for installation, configuration, and evaluation at `https://github.com/coralabo/ResKV`.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The research focuses on algorithm optimization for LLM KV cache pruning and contains no crowdsourcing tasks, human participant experiments, or related data collection processes.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work conducts pure algorithmic and computational experiments without engaging human participants or human-related research, so no participant risk disclosure or IRB approval is needed.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: No LLMs were adopted as core, original, or non-standard components for method design, experimentation, or technical innovation in this work. Minor LLM-assisted writing and polishing do not require formal declaration.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.