POSITION-AWARE COOPERATIVE TOKEN-CHANNEL COMPRESSION WITH KEY-VALUE DECOUPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

The linearly growing key-value (KV) cache imposes a critical storage bottleneck for large language models (LLMs) in long-context inference scenarios, hindering their practical deployment. Existing methods compress the KV cache by evicting or merging tokens and reducing channel redundancy, thereby reducing its memory footprint. However, these merging methods overlook that after Rotary Position Embedding (RoPE), semantically similar tokens can yield inconsistent attention responses due to positional differences, whereas tokens with similar attention responses have higher positional-function similarity and are more mergeable. Moreover, these methods handle the token and channel dimensions in isolation, and apply a uniform compression strategy to both K and V, overlooking token-channel cooperative potential and distinct roles of K and V in the attention mechanism. In this paper, we propose a Position-Aware Cooperative Token-Channel Compression with Key-Value Decoupling method (PACT-KV). It designs positional-function similarity to measure the mergeability of tokens within the SimHash cluster for K-side compression, which jointly considers semantics and attentions, and avoids positional interference brought by physical proximity. Meanwhile we propose a cluster-conditioned K-side channel pruning to capture consistent channel-energy patterns in the same cluster, which naturally cooperates with the K-side token merging to reduce the overhead. Due to the semantic aggregation role of V, which contrasts with the query-matching role of K, PACT-KV further performs an independent V-side token merging and executes V-side channel pruning when V tokens still exceed the budget. Experimental results on LongBench and Needle-in-a-Haystack (NIAH) show that PACT-KV achieves strong performance while significantly reducing both memory footprint and inference latency.

1 INTRODUCTION

LLMs are rapidly evolving toward advanced scenarios (Zhao et al., 2026), such as complex reasoning (Wei et al., 2022) and long-context understanding (Kuratov et al., 2024). To support efficient autoregressive generation, LLMs heavily rely on the KV cache to avoid redundant computation over historical token states (Shazeer, 2019). However, the memory footprint of the KV cache grows linearly with the context length and batch size, creating substantial memory pressure (Kwon et al., 2023) and limiting decode-stage throughput and latency in long-context scenarios (Pope et al., 2023). Therefore, efficiently compressing the KV cache without sacrificing output quality has become a core bottleneck in long-context inference optimization.

Training-free KV cache compression operates along the token and channel dimensions. For the token side, eviction methods such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024b) discard tokens based on attention importance, risking information loss under high compression ratios. Merging methods such as CaM (Zhang et al., 2024) integrate evicted tokens into retained states, and KeepKV (Tian et al., 2026) further compensates for merging-induced attention changes. However, raw semantic similarity, attention importance, or positional proximity alone may not capture how semantically similar tokens interact with queries after RoPE. Specifically, due to the RoPE operation (Su et al., 2024), tokens with similar pre-RoPE K representations can produce different attention responses—the vectors of attention weights induced by the same query sequence. By contrast, tokens that yield similar attention responses, which jointly reflect semantics, attention distribution, and positional information, are more likely to share similar positional functions, making them better candidates for merging. For the channel side, structured pruning (Xu et al., 2025) uses coarse-grained

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

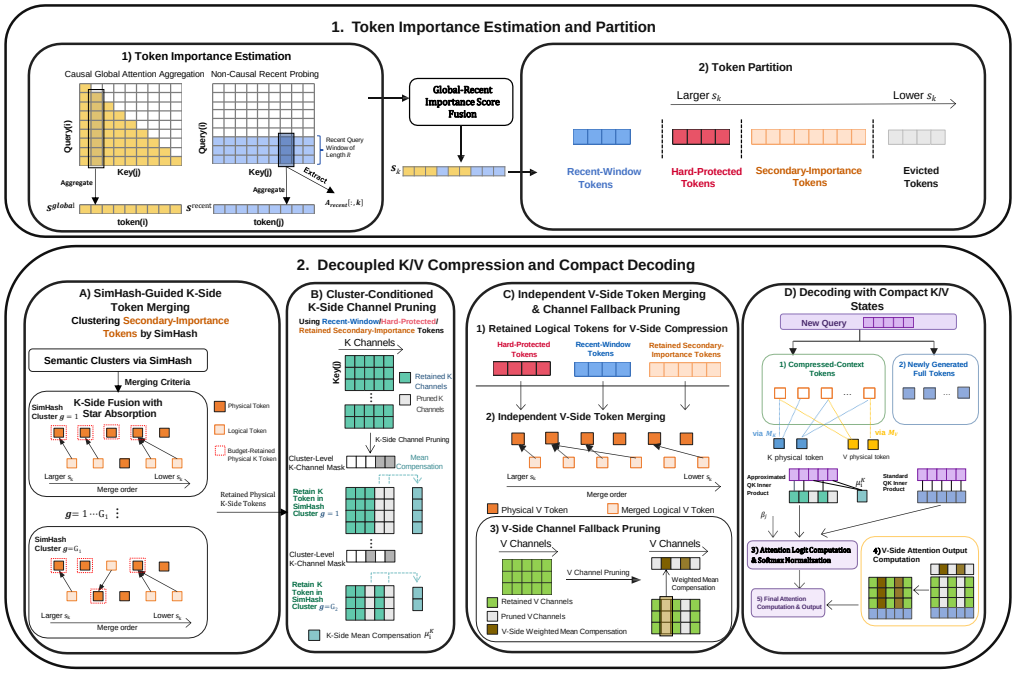


Figure 1: The framework of PACT-KV. Causal global attention and non-causal recent probing guide token partitioning into recent-window, hard-protected, secondary-importance, and evicted sets. K-side compression combines SimHash-guided merging of secondary-importance tokens with cluster-conditioned channel pruning. V-side compression independently merges retained logical tokens, followed by channel fallback pruning if the physical token count still exceeds the budget. Decoding uses compact K/V states through independent logical mappings without explicit reconstruction.

masks that may miss region-specific channel preferences, whereas unstructured pruning (Liao et al., 2026) offers finer granularity at greater computation and recovery overhead.

In addition, existing methods often treat token and channel compression separately and overlook the asymmetric redundancy between K and V due to their distinct roles in learning attention outputs. K supports query matching, whereas V supports semantic aggregation, implying different compression tolerances. A uniform K/V compression strategy can therefore be limited by the more sensitive side, motivating decoupled K/V compression with complementary token- and channel-level operations.

In this paper, we propose PACT-KV for position-aware cooperative token-channel compression with K/V decoupling. As shown in Fig. 1, we first combine non-causal local and causal global attention to estimate token importance, then perform K/V compression separately. On the K side, we jointly model key-semantic clusters and positional-function differences for token merging, and propose a deployment-friendly cluster-level channel pruning paradigm. First, we anchor key-semantic clusters by applying SimHash (Charikar, 2002) to pre-RoPE K features. This not only provides a safe domain for K-side token merging, but also naturally motivates cluster-conditioned K-side channel pruning, achieving a compromise between global masks and per-token masks. Second, to handle positional-function differences among tokens, we compute recent-query attention responses for tokens within the same raw-K semantic cluster, and take these attention responses as positional-function signatures to examine whether the tokens exhibit similar positional function during K-side token merging. Meanwhile, to prevent overly dispersed semantic representations from hindering subsequent V-side token merging, we additionally require K-side merging candidates to satisfy a V-space cosine-similarity constraint. After K-side token merging, PACT-KV further performs structured K-channel pruning within each SimHash cluster by estimating channel importance from the weighted absolute QK interaction between recent queries and retained keys, allowing each cluster to retain its own important channels. On the V side, we perform fully decoupled token-level merging on V, releasing token budget first and then performing channel pruning if the budget is still exceeded. The main contributions are summarized as follows:

- We propose a positional-function-aware K-side token merging strategy based on SimHash-guided key-semantic clusters and recent-query attention-response similarity. By constrain-

ing raw semantic similarity while distinguishing positional-function differences, this strategy substantially improves the reliability and fidelity of K-side token merging.

- We develop cluster-conditioned K-side channel pruning that exploits within-cluster channel-saliency consistency while preserving between-cluster differences, balancing selection granularity and implementation complexity without the high overhead of fine-grained unstructured pruning.
- We design an asymmetric K/V cache compression framework that coordinates token-level merging and channel-level pruning under a unified memory budget. Through KV decoupling, the framework maximizes the complementary compression potential of K and V in their respective representation spaces.
- We compare PACT-KV with state-of-the-art methods on LongBench and NIAH, and evaluate its runtime and memory efficiency. PACT-KV achieves the best average score across all tested budget settings. Efficiency results further show that PACT-KV reduces peak memory compared with full-cache inference and maintains high steady-state decoding efficiency.

2 Related Work

Channel-dimensional methods reduce memory overhead by pruning redundant feature channels. ThinK (Xu et al., 2025) performs query-driven structured K-channel pruning. LeanK (Zhang et al., 2025) learns static K-cache masks. SPARK (Liao et al., 2026) improves pruning granularity through token-wise unstructured pruning with recoverable reconstruction. These methods reveal channel-level redundancy, but mainly place compression pressure on the channel dimension. By contrast, PACT-KV performs structured K-channel pruning within semantically similar token clusters and coordinates token- and channel-level compression under KV decoupling.

Token eviction methods reduce cache length by retaining only selected KV entries. StreamingLLM (Xiao et al., 2024) preserves attention sinks and recent tokens for fixed-size streaming inference. H2O (Zhang et al., 2023) and Scissorhands (Liu et al., 2023) retain tokens according to accumulated or persistent attention importance, while SnapKV (Li et al., 2024b) selects important prompt tokens from an observation window before generation. Recent studies further consider inter-layer and inter-head structural heterogeneity. Layer-wise methods such as PyramidKV (Cai et al., 2024) and SqueezeAttention (Wang et al., 2025) allocate different cache budgets according to layer-level attention characteristics, while head-wise methods such as AdaKV (Feng et al., 2025) and DuoAttention (Xiao et al., 2025) exploit functional differences among attention heads for head-aware cache management. Such methods mainly schedule the cache budget from the perspective of cross-layer or cross-head heterogeneity, which is orthogonal to compression methods under a unified budget.

Token merging methods reduce cache length by transferring information from less important tokens to retained KV entries instead of directly discarding them. SemantiCache (Wu et al., 2026) relies on semantic chunking and cluster merging. EMS (Li et al., 2024a) uses global-local importance with KV cosine similarity. KVMerger (Wang et al., 2024) performs model-guided adaptive KV merging. CaM (Zhang et al., 2024) merges adjacent KV pairs by key similarity and weighted value averaging. KeepKV (Tian et al., 2026) records merging history and compensates for merging-induced changes in attention through its Electoral Votes mechanism and ZIP-Merging. Existing merging strategies mostly rely heavily on shallow semantic continuity or attention scores, and do not explicitly distinguish positional-function independence under the same semantics. In contrast, this paper introduces recent-query attention-response similarity within SimHash-guided key-semantic clusters, reducing the risk of incorrectly merging tokens with different positional functions.

3 Methodology

We first review the standard KV cache computation, focusing on a single attention layer and head with channel dimension D. During the prefill stage, an initial prompt of length L is mapped to the global key matrix $K_{\leq L}$ and value matrix $V_{\leq L}$, which are stored in GPU memory as the initial KV cache. The model then enters the decode stage. For the newly generated token at step $L{+}m$ ($m \geq 1$), the mapped query, key, and value vectors are denoted by q_{L+m}, k_{L+m}, $v_{L+m} \in \mathbb{R}^D$, respectively. To avoid redundant computation over historical states, the current k_{L+m} and v_{L+m} are autoregressively appended to the KV cache: $K_{\leq L+m} = [K_{\leq L+m-1}; k_{L+m}], \quad V_{\leq L+m} = [V_{\leq L+m-1}; v_{L+m}]$.

The attention output at the current step is then computed by applying the new query to the complete historical KV cache: $O_{L+m} = \mathrm{Softmax}_{\mathrm{dim}=-1}\left(\frac{q_{L+m}K^T_{\leq L+m}}{\sqrt{D}}\right)V_{\leq L+m}$.

3.1 Token Importance Estimation and Partition

Combining global accumulated attention with local (recent-window) attention to overcome single-view bias has become a mature paradigm for token importance estimation. However, to mitigate the bias in positional importance estimation due to the causal mask constraining traditional local attention scores, we propose a Non-Causal Recent Probing method for local importance estimation. This probing is used only for compression and does not alter causal attention during generation.

For the local importance score s_k^{recent}, queries from a recent window of length R, denoted by $Q_{\mathrm{recent}} \in \mathbb{R}^{R\times D}$, attend to all historical keys $K_{1:L}$ without a causal mask:

$$s_k^{\mathrm{recent}} = \sum_{\mathrm{row}=1}^{R} A_{\mathrm{recent}}[\mathrm{row}, k], A_{\mathrm{recent}} = \mathrm{Softmax}\left(\frac{Q_{\mathrm{recent}}K^T_{1:L}}{\sqrt{D}}\right) \in \mathbb{R}^{R\times L}, \tag{1}$$

where A_{recent} is named the "recent-query attention-response matrix". The k-th column $A_{\mathrm{recent}}[:, k]$ represents the recent-query *attention response* of token k. This provides a basis for constructing positional-function signatures for subsequent merging.

For the global importance score s_k^{global}, we retain standard causal attention aggregation to characterize historical accumulated attention:

$$s_k^{\mathrm{global}} = \sum_{t=k}^{L} \mathrm{Softmax}\left(\frac{Q_t K^T_{1:t}}{\sqrt{D}}\right)_k, \tag{2}$$

where the subscript k is to extract the k-th element of the vector $\mathrm{Softmax}\left(\frac{Q_t K^T_{1:t}}{\sqrt{D}}\right)$.

To align score scales, we rescale the global score by the total-mass ratio and take its element-wise maximum with the recent score, obtaining the integrated importance score $s_k (k = 1, \cdots, L)$:

$$s_k = \max\left(s_k^{\mathrm{global}} \cdot \frac{\sum_i s_i^{\mathrm{recent}}}{\sum_i s_i^{\mathrm{global}} + \epsilon},\ s_k^{\mathrm{recent}}\right). \tag{3}$$

To prevent globally redundant tokens from diluting core anchor importance and noisy tokens from biasing merging, we partition historical tokens into three categories. The most important class, accounting for 10% of the budget, is hard-protected together with recent-window tokens against K-side merging and budget-driven eviction. The low-importance class, accounting for 20% of the prompt, is treated as pure noise and removed. The remaining tokens of secondary importance are the sole candidates for K-side token merging.

3.2 SimHash-Guided K-Side Token Merging and Channel Pruning

3.2.1 Hash-Based Semantic Clustering

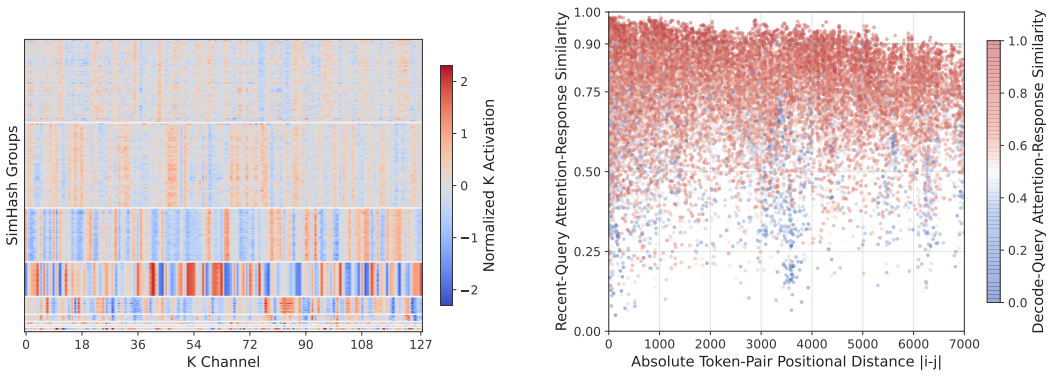
Existing KV cache compression methods that select anchors or merge tokens solely by importance scores can concentrate retained tokens in certain semantic regions. To avoid semantic loss, we group semantically similar tokens and merge within each group.

We use SimHash, a lightweight locality-sensitive hashing method for similarity-based grouping (Ravichandran et al., 2005), to cluster key states. To avoid RoPE phase interference, we hash pre-RoPE keys K_{raw} using a fixed random projection matrix $R_{\mathrm{hash}} \in \mathbb{R}^{D\times C}$. With C hash bits, token k's bucket ID is computed as:

$$H = K_{\mathrm{raw}}R_{\mathrm{hash}}, \quad g(k) = \sum_{c=1}^{C} \mathbb{I}(H_{k,c} > 0) \cdot 2^{c-1}. \tag{4}$$

The resulting bucket (cluster) ID $g(k)$ is then used to strictly constrain subsequent K-side token merging within the same hash bucket.

We visualize the channel-wise Z-score-standardized pre-RoPE K activations of tokens grouped by SimHash. As shown in Fig. 2(a), tokens in the same SimHash cluster exhibit highly consistent standardized K-activation patterns, whereas different clusters show distinct channel preferences. This observation suggests that SimHash provides an effective semantic partition. We therefore use SimHash clusters to guide K-side token merging and structured channel pruning within each cluster.



(a) (b)

Figure 2: Empirical analysis of SimHash clustering and attention-response similarity, conducted on LongBench with LLaMA3-8B-Instruct. (a) Normalized K-activation patterns of tokens in the top-10 largest SimHash clusters obtained with 8 hash bits. (b) Pairwise positional distance, recent-query attention-response similarity, and decode-query attention-response similarity for token pairs with raw-K cosine similarity above 0.9. Recent-query attention-response vectors are obtained from the attention distributions induced by 24 recent probing queries. Each point denotes a token pair, colored by attention-response similarity measured from queries collected during greedy decoding.

3.2.2 SimHash-Guided K-Side Token Merging

Merging Criteria Although same-bucket tokens have high K-based semantic similarity, blind merging can still introduce large errors due to positional encoding and query interaction. To reduce attention-logit shifts from K-side merging, candidate pairs should exhibit similar attention responses under subsequent queries.

However, real decode queries are unavailable during prefill, so a computable proxy is needed. To this end, we analyze the relationship among token-pair distance, recent-query attention-response similarity, and real decode-query attention-response similarity. As shown in Fig. 2(b), many distant token pairs still show high attention-response similarity under decode queries. This suggests that merging based on physical distance may miss long-range mergeable pairs and introduce incorrect local merges. In contrast, token pairs with high recent-query attention-response similarity are mostly colored red, indicating that recent-query attention-response is more consistent with the attention distribution under real decoding queries. Therefore, to assess positional-function similarity between tokens, we use the recent-query attention response of token k as its positional-function signature, denoted as $w_k = A_{\text{recent}}[:, k]$. Meanwhile, since K-side merging does not merge the corresponding V states, we impose a V-space cosine-similarity constraint to avoid excessive dispersion before later V-side merging. Finally, same-bucket tokens i and j are mergeable only if both constraints hold:

$$\text{CosSim}(w_i, w_j) > \tau_w \quad \wedge \quad \text{CosSim}(V_i, V_j) > \tau_v, \tag{5}$$

where CosSim denotes cosine similarity, and V_i is the i-th token's V feature.

At this point, mergeability between tokens i and j is checked at three levels: SimHash clustering provides coarse-grained semantic boundaries, τ_w ensures positional-function similarity, and the relatively mild τ_v preserves V-side consistency to avoid potential obstacles to merging.

K-Side Fusion with Star Absorption Within each bucket, tokens are scanned in descending order of s_k. Each unmerged token i becomes a physical anchor and absorbs all lower-score unmerged tokens satisfying Eq. (5). These absorbed tokens are logical tokens mapped to i on the K side, and their set is denoted by $\mathcal{C}_i$. The resulting star-shaped groups could avoid error accumulation from chained merging.

For an absorbed token j that loses its independent K feature, we record its attention quality relative to the physical token. We use the ratio between integrated importance scores to approximate their relative contributions in the unnormalized attention-score space, and directly transform it into an additive logarithmic bias in the Softmax logit space:

$$\beta_j = \log \rho_{j \to i} = \log \frac{s_j}{s_i}. \tag{6}$$

After physical token i absorbs its logical tokens and β_j is computed, it accumulates the scores of the absorbed logical tokens to update its importance score:

$$s_i \leftarrow s_i + \sum_{j \in \mathcal{C}_i} s_j. \tag{7}$$

Finally, we rank physical tokens by their updated scores in Eq. (7) and retain the top-ranked key states allowed by the budget, together with logical-token additive biases. Associated physical and logical V tokens are preserved. During reconstruction, logical token j uses its mapped physical token i's key feature, with β_j added to its attention logit.

3.2.3 Cluster-Conditioned K-Side Channel Pruning

As discussed above, the K features after SimHash clustering exhibit a significant cluster-specific pattern in channel energy distributions. In the channel dimension, a globally shared channel mask overlooks token-specific feature differences, whereas per-token masks ignore within-cluster regularities and incur unstructured processing overhead. We therefore propose K-Side Channel Pruning based on SimHash clustering, sharing a mask within each bucket while retaining cluster-specific channels.

For the retained physical token set $\mathcal{P}_K^{(g)}$ in cluster g, we measure channel importance through per-channel QK interactions with recent queries, using absolute values to prevent sign cancellation and taking the updated importance scores s_i as weights:

$$E_c^{K,g} = \sum_{i \in \mathcal{P}_K^{(g)}} s_i \cdot |K_{i,c}| \cdot \sum_{t=L-R+1}^{L} |Q_{t,c}|, \tag{8}$$

where $E_c^{K,g}$ is importance score of the c-th K channel in cluster g, $K_{i,c}$ and $Q_{t,c}$ are the c-th channel values of the key state of physical token i and recent query Q_t, respectively. Then channels within each cluster are sorted in descending order according to the channel energy $E_c^{K,g}$ and truncated under a unified pruning rate ρ_K, yielding a pruned channel set $\Omega_{\text{prune}}^{K,g}$ and a retained channel set $\Omega_{\text{keep}}^{K,g}$ of cluster g.

In addition, to reduce the logit shift caused by K-side channel pruning, we avoid zero-filling the pruned channels. For the pruned channel set $\Omega_{\text{prune}}^{K,g}$ of cluster g, we extract a scalar mean compensation term for each physical token i:

$$\mu_i^K = \frac{1}{|\Omega_{\text{prune}}^{K,g}|} \sum_{c \in \Omega_{\text{prune}}^{K,g}} K_{i,c}. \tag{9}$$

This term is later used in reconstruction stage as an energy approximation for the pruned channels.

3.3 Independent V-Side Token Merging and Channel Fallback Pruning

Independent V-Side Token Merging Unlike the K side, which dominates attention logits, the V side is responsible only for semantic aggregation under given attention weights. Suppose a V-side fusion group $\mathcal{G}_i^V$ is fused into a unified physical representation V_i^{merge}. Given K-side weights $a_j \geq 0$, the output contributions before and after fusion are $O_i^V = \sum_{j \in \mathcal{G}_i^V} a_j V_j$ and $\tilde{O}_i^V = \sum_{j \in \mathcal{G}_i^V} a_j V_i^{\text{merge}}$, respectively. The local semantic aggregation error introduced by this fusion satisfies the following upper bound:

$$\left\| O_i^V - \tilde{O}_i^V \right\| \leq \left(\sum_{j \in \mathcal{G}_i^V} a_j \right) \max_{j \in \mathcal{G}_i^V} \|V_j - V_i^{\text{merge}}\|. \tag{10}$$

Eq. (10) derived in Appendix A, shows that, given attention weights, the V-side fusion error mainly depends on representation consistency in the value space. Therefore, PACT-KV merges retained logical tokens based on V-side content similarity, independently of K-side hash buckets and token merging.

The V-side merging follows the same Star Absorption order as the K side, while using only V-side cosine similarity as the merging criterion:

$$\text{CosSim}(V_i, V_j) > \tau_v. \tag{11}$$

After absorption, the center physical V is updated by averaging the merged V states weighted by their integrated importance scores and shared by the merged logical tokens.

V-Side Channel Fallback Pruning If the fused physical V token count still exceeds the budget, we apply V-side channel fallback pruning. Let $\mathcal{P}_V$ denote the retained fused physical V token set. Since V states neither determine attention logits nor undergo RoPE rotation, we score channels by their globally weighted absolute magnitudes:

$$E_c^V = \sum_{i \in \mathcal{P}_V} s_i \cdot |V_{i,c}| \,. \tag{12}$$

Under a globally shared mask, high-score channels form Ω^V_{keep} and the remaining channels form Ω^V_{prune}. We compute a global weighted mean compensation vector $\mu^V_{\Omega^V_{\text{prune}}}$ for the pruned channels:

$$\mu^V_{\Omega^V_{\text{prune}}} = \frac{\sum_{i \in \mathcal{P}_V} s_i \cdot V_{i,\Omega^V_{\text{prune}}}}{\sum_{i \in \mathcal{P}_V} s_i}. \tag{13}$$

This vector is stored as a recovery parameter and later used during decoding to approximate the aggregated contribution of the pruned V channels.

3.4 Decoding with Compact K/V States

During decoding, PACT-KV operates directly on compact physical states through independent K/V mappings (M_K and M_V), without explicitly reconstructing full K/V states. Let $\mathcal{C}_{\text{ctx}}$ and $\mathcal{C}_{\text{gen}}$ denote retained context logical tokens and newly generated tokens, respectively. The mappings $M_K(j)$ and $M_V(j)$ identify the corresponding physical K and V tokens for logical token j.

For query $q_{\text{new}} \in \mathbb{R}^D$, let $\widetilde{K}_{\text{ctx}}, \widetilde{V}_{\text{ctx}} \in \mathbb{R}^{|\mathcal{C}_{\text{ctx}}| \times D}$ denote the approximate reconstructed states in logical token order. For logical token j, let $i = M_K(j)$ belong to SimHash cluster g. For channel $c \in \{1, \dots, D\}$, define

$$\widetilde{K}_{\text{ctx}}[j,c] = \begin{cases} K_{i,c}, & c \in \Omega^{K,g}_{\text{keep}}, \\ \mu^K_i, & c \in \Omega^{K,g}_{\text{prune}}, \end{cases} \qquad \widetilde{V}_{\text{ctx}}[j,c] = \begin{cases} V_{M_V(j),c}, & c \in \Omega^V_{\text{keep}}, \\ \mu^V_c, & c \in \Omega^V_{\text{prune}}. \end{cases} \tag{14}$$

Here, $[j,c]$ indexes logical token j and channel c; $V_{M_V(j),c}$ refers to the merged physical V state, and μ^V_c denotes channel c of the compensation vector $\mu^V_{\Omega^V_{\text{prune}}}$. Compensation is unused when no channels are pruned.

Let $K_{\text{gen}}, V_{\text{gen}} \in \mathbb{R}^{|\mathcal{C}_{\text{gen}}| \times D}$ denote uncompressed generated-token states, and let $\boldsymbol{\beta} = [\beta_j]_{j \in \mathcal{C}_{\text{ctx}}}$ contain K-side logarithmic biases in the same logical token order, with zero bias for tokens that are not redirected. The attention output is

$$O_{\text{new}} = \text{Softmax}\left(\left[\frac{q_{\text{new}} \widetilde{K}^T_{\text{ctx}}}{\sqrt{D}} + \boldsymbol{\beta}, \frac{q_{\text{new}} K^T_{\text{gen}}}{\sqrt{D}} \right] \right) \begin{bmatrix} \widetilde{V}_{\text{ctx}} \\ V_{\text{gen}} \end{bmatrix}. \tag{15}$$

The softmax jointly normalizes over context and generated tokens. Attention is computed directly from compact states without explicit K/V reconstruction, as detailed in Appendix B.

4 Experiments

4.1 Experimental Setup

We evaluate PACT-KV on LongBench (Bai et al., 2024) and Needle-in-a-Haystack (NIAH) (Kamradt, 2023). For models, we use LLaMA3-8B-Instruct and LLaMA3.1-8B-Instruct (Team, 2024), as well as Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). We compare PACT-KV with strong token-level baselines such as StreamingLLM (Xiao et al., 2024), H2O (Zhang et al., 2023), CaM (Zhang et al., 2024), SnapKV (Li et al., 2024b), KeepKV (Tian et al., 2026), and the combinations of token-level and channel-level methods, i.e., SnapKV or H2O combined with SPARK (Liao et al., 2026) or ThinK (Xu et al., 2025). The methods are evaluated by the built-in metrics of the datasets.

All experiments can be conducted on a single NVIDIA A100 GPU with 40GB memory. Unless otherwise specified, the budget used below refers to the KV cache memory required to store the given number of full-channel KV token states. Thus, using the same budget ensures the same equivalent KV cache memory constraint across models. By default, PACT-KV uses the following hyperparameters: $C = 7$, $\tau_v = 0.7$, $\tau_w = 0.95$, $\rho_K = 0.3$, and $R = 24$. Hyperparameter sensitivity analysis is provided in Appendix C. For fair comparison, we use the same recent window size as SnapKV, SPARK, and ThinK, and adopt fixed per-layer budgets for KeepKV, with all other settings following its official implementation defaults.

Table 1: LongBench performance under an equivalent KV cache budget of 256.

	Method	Single-Doc			Multi-Doc			Summ.			Few-shot			Synth.		Code		Avg.
		NQA	Qasp	MF	HQA	2Wiki	Musq	GovR	QM	MN	TREC	Triv	SAM	PCnt	PRe	Lcc	RB-P	
Llama-3-8B-Instruct	Full KV	22.43	43.85	44.16	46.70	36.78	21.13	30.13	22.60	27.78	75.00	90.48	42.41	5.50	64.50	59.23	54.26	42.93
	KV Cache Size = 256																	
	StreamingLLM	20.07	26.79	29.52	41.04	31.12	17.05	19.19	20.03	20.30	67.50	79.19	39.05	6.00	62.50	56.61	52.60	36.79
	H2O	20.69	32.35	35.15	44.32	31.92	18.92	23.20	21.12	25.31	72.00	86.45	38.61	5.00	56.50	57.55	54.43	38.97
	CaM	18.79	24.86	31.06	41.69	29.16	18.26	20.08	19.26	21.68	69.00	75.02	35.77	4.00	57.00	54.30	51.25	35.70
	SnapKV	21.85	37.40	**41.93**	44.39	35.00	20.15	20.27	20.39	24.10	73.00	80.55	**40.90**	5.50	**66.50**	56.05	49.12	39.82
	KeepKV	20.05	30.70	37.22	44.61	33.93	**21.85**	21.55	21.28	22.35	68.50	89.82	40.42	5.00	62.50	**59.03**	**56.81**	39.73
	PACT-KV	**23.01**	**38.90**	41.01	**45.98**	**35.83**	21.57	**24.92**	**22.33**	**26.16**	**74.00**	**89.97**	40.57	**7.25**	60.50	58.22	49.45	**41.23**
Llama-3.1-8B-Instruct	Full KV	28.18	45.53	54.85	56.52	46.66	31.20	35.18	25.36	27.21	74.00	91.65	43.82	8.38	99.50	65.16	58.60	49.49
	KV Cache Size = 256																	
	StreamingLLM	20.60	23.15	32.08	46.77	37.70	24.36	20.09	20.96	19.27	65.00	87.50	41.24	7.64	**99.50**	61.24	51.13	41.14
	H2O	25.37	31.62	40.47	52.55	43.69	28.01	27.69	22.42	25.45	70.00	90.14	40.71	7.26	99.00	61.27	53.50	44.95
	CaM	21.97	21.79	35.46	46.54	41.09	24.51	21.48	20.34	19.93	66.00	81.78	38.62	7.84	99.00	58.61	50.66	40.98
	SnapKV	24.85	35.15	50.65	54.94	45.16	28.88	24.65	23.74	23.13	**73.00**	91.27	41.07	**8.91**	**99.50**	61.41	**55.18**	46.34
	KeepKV	24.94	25.15	42.87	50.14	41.33	27.95	23.31	23.26	22.49	68.00	91.51	41.60	6.50	99.00	61.66	54.28	44.00
	PACT-KV	**28.58**	**36.69**	**52.33**	**55.48**	**45.23**	**29.64**	**28.99**	**23.99**	**26.02**	72.50	**92.31**	**42.98**	7.69	**99.50**	**61.80**	53.46	**47.32**
Mistral-7B-Instruct	Full KV	29.14	38.27	50.10	51.54	36.16	26.67	34.16	25.86	26.49	76.00	88.59	47.48	5.50	97.00	60.44	61.64	47.19
	KV Cache Size = 256																	
	StreamingLLM	20.32	20.31	27.32	37.74	26.79	16.19	19.35	19.76	18.40	66.50	85.37	42.95	**7.50**	51.00	56.75	54.16	35.65
	H2O	22.90	29.09	37.00	46.03	30.51	21.99	27.48	22.06	**25.51**	71.00	86.89	43.14	5.00	55.00	57.61	53.99	39.70
	CaM	11.30	18.57	29.09	29.01	30.41	13.07	16.81	19.17	20.01	65.00	81.53	34.47	2.97	4.00	46.22	51.27	29.56
	SnapKV	25.42	30.47	**46.43**	47.31	31.73	21.59	24.13	23.08	23.06	**74.50**	**89.13**	43.28	3.50	**95.50**	57.73	57.76	43.41
	KeepKV	22.12	26.22	38.31	41.25	30.01	19.31	23.97	23.30	22.40	63.50	87.86	44.94	4.00	88.50	**58.51**	**58.09**	40.77
	PACT-KV	**25.94**	**34.53**	46.36	**49.12**	**32.59**	**23.47**	**28.25**	**24.31**	25.07	73.50	89.06	**44.97**	5.50	92.50	57.36	56.91	**44.34**

Table 2: Comparison with SPARK- and ThinK-enhanced token-level baselines on LongBench category averages using Llama-3-8B-Instruct under an equivalent KV cache budget of 256.

Method	Single-Doc	Multi-Doc	Summ.	Few-shot	Synth.	Code	Avg.
SPARK+H2O	28.99	33.13	22.24	66.53	33.00	56.00	39.42
SPARK+SnapKV	34.00	33.29	22.26	65.74	**36.25**	52.31	40.19
ThinK+H2O	28.95	33.49	21.54	65.46	32.75	58.53	39.43
ThinK+SnapKV	33.36	33.97	21.17	67.10	34.75	**59.05**	40.90
PACT-KV	**34.31**	**34.46**	**24.47**	**68.18**	33.88	53.84	**41.23**

4.2 Performance on LongBench

LongBench covers six long-context task categories. We report results at an equivalent budget of 256 using default hyperparameters.

As shown in Table 1, PACT-KV achieves the highest average LongBench score among compression baselines on all three models at a budget of 256, indicating better preservation of task-relevant information under high compression ratios.

We further use SPARK and ThinK, two channel pruning methods, together with token-level methods H2O and SnapKV as additional baselines. As shown in Table 2, SPARK and ThinK improve these token-level methods, confirming the benefit of channel pruning. Nevertheless, the combined methods still underperform PACT-KV, indicating that simple stacking cannot fully exploit token-channel complementarity under the same equivalent budget.

We further evaluate PACT-KV under a tighter equivalent KV cache budget of 128, where it achieves the highest average LongBench score on all three tested models. Complete results are provided in Appendix D.

4.3 Performance on Needle-in-a-Haystack

Needle-in-a-Haystack evaluates information retrieval from long noisy contexts. We test LLaMA3.1-8B-Instruct at context lengths from 8K to 32K with document depth intervals of 10%. Table 3 shows that PACT-KV significantly outperforms all baselines across context lengths and insertion depths, demonstrating robust preservation of distant key information under KV cache compression.

4.4 Ablation Study

We conduct ablation studies on LLaMA3-8B-Instruct using five representative LongBench tasks: Multi-News, LCC, NarrativeQA, Passage Count, and QMSum. Except the ablated parts, all components use the default hyperparameters and an equivalent budget of 256.

Table 3: Needle-in-a-Haystack average score under different KV cache budgets.

Budget	H2O	CaM	SnapKV	KeepKV	PACT-KV
64	22.82	28.59	59.25	55.34	**71.31**
96	30.63	40.65	62.86	56.85	**77.01**
128	31.46	41.32	67.07	56.27	**80.34**

Table 4: Average ablation results of PACT-KV under a KV cache budget of 256.

Variant	Avg.
PACT-KV	**28.66**
w/o Secondary-Importance Constraint	25.92
w/o SimHash Clustering	23.66
w/o Token Merging	25.81
w/o Channel Pruning	26.63
w/o Token Merging and Channel Pruning	27.65
w/o Token–Channel Complementarity	24.13

We evaluate variants with individual components removed or modified. To ablate token–channel complementarity, we retain the same compression operations but apply token merging and channel pruning sequentially at the same compression ratio, without cross-dimensional coordination.

As shown in Table 4, removing token merging or channel pruning reduces the average score from 28.66 to 25.81 or 26.63, respectively. Disabling token–channel coordination further reduces it to 24.13, supporting the importance of their complementary interaction.

Removing the secondary-importance constraint yields 25.92, while removing SimHash clustering causes the largest drop, to 23.66, highlighting the importance of semantic grouping for token merging and structured channel pruning.

Table 5: Runtime (TTFT in s, TPOT in ms), E2E throughput (token/s), and GPU memory (GiB) on a single RTX 3090. TTFT includes prefill and the first decode step; TPOT excludes that step.

		Decode = 128				Decode = 1K			
Prompt	Method	TTFT	TPOT	E2E	Memory	TTFT	TPOT	E2E	Memory
4K	H2O	1.937	49.37	15.56	16.29	2.062	47.79	20.03	16.51
	CaM	2.190	46.80	15.61	15.91	2.172	49.21	19.32	15.91
	KeepKV	1.172	88.48	10.36	15.79	1.175	87.22	11.31	15.79
	PACT-KV	3.124	32.24	17.09	15.93	3.174	34.99	25.77	16.04
8K	H2O	4.982	48.27	11.49	17.27	4.994	57.14	16.09	17.54
	CaM	5.626	50.41	10.58	16.41	5.498	54.03	16.71	16.46
	KeepKV	2.377	87.91	9.49	16.23	2.373	88.03	11.06	16.27
	PACT-KV	6.396	32.98	11.78	16.39	6.392	35.54	23.53	16.50

4.5 Efficiency Evaluation

To evaluate practical efficiency, we measure runtime and GPU memory usage using LLaMA3-8B-Instruct on a single RTX 3090 GPU. For fair comparison, we mainly focus on token merging methods and omit token eviction methods. Except for full-cache inference, all compression methods use an equivalent budget of 256. As shown in Table 5, despite higher TTFT from token- and channel-dimensional compression, PACT-KV improves end-to-end throughput over the baselines even with a short decode length of 128 tokens, with larger gains at 1K tokens. It also reduces memory usage compared with full-cache inference and remains executable at 8K input length, where full-cache inference runs out of memory.

5 Conclusion

We propose PACT-KV, a KV cache compression method combining positional-function-aware token merging with K/V-asymmetric token-channel compression. SimHash-guided grouping and recent-query attention responses guide K-side merging, while cluster-conditioned K-channel pruning and independent V-side compression exploit their distinct redundancy patterns. Experiments demonstrate strong performance after compression, reduced memory usage, and high steady-state decoding efficiency in long-context inference.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language translation, language editing, and refinement of academic expression, including improvements to the grammar, wording, readability, and coherence of selected sentences. These tools also provided suggestions on the organization and presentation of parts of the manuscript.

Generative AI tools were not used to generate experimental data, conduct experiments in place of the authors, independently draw research conclusions, or provide interpretations of experimental results without verification by the authors. The core research ideas, method design, experimental design, experimental results, and final conclusions were developed or obtained and confirmed by the authors.

All content generated or modified with generative AI assistance was manually reviewed, revised, and verified by the authors to ensure consistency with the original research and the accuracy of the technical descriptions, mathematical expressions, and scholarly statements. AI-generated suggestions served only as supporting input; the authors independently evaluated and approved all content ultimately incorporated into the manuscript.

The authors take full responsibility for the final content of this paper, including all text, arguments, and other material produced or modified with generative AI assistance.

REFERENCES

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

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic KV cache compression based on pyramidal information funneling. *CoRR*, abs/2406.02069, 2024.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Annual ACM Symposium on Theory of Computing*, pp. 380–388, 2002.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

Greg Kamradt. Needle in a haystack – pressure testing llms. GitHub repository, 2023. [Online]. Available: https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Y. Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. In *Conference on Neural Information Processing System (NeurIPS)*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023.

Yingxin Li, Ye Li, Yuan Meng, Xinzhu Ma, Zihan Geng, Shutao Xia, and Zhi Wang. EMS: adaptive evict-then-merge strategy for head-wise KV cache compression based on global-local importance. *CoRR*, abs/2412.08521, 2024a.

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 *Conference on Neural Information Processing Systems (NeurIPS)*, 2024b.

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. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 31961–31969, 2026.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *International Conference on Machine Learning (ICML)*, 2023.

Deepak Ravichandran, Patrick Pantel, and Eduard H. Hovy. Randomized algorithms and NLP: using locality sensitive hash functions for high speed noun clustering. In *43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 622–629, 2005.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. Keepkv: Achieving periodic lossless kv cache compression for efficient llm inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33259–33267, 2026.

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. *CoRR*, abs/2407.08454, 2024.

Zihao Wang, Bin Cui, and Shaoduo Gan. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. In *International Conference on Learning Representations (ICLR)*, pp. 23593–23606, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Shunlong Wu, Hai Lin, Shaoshen Chen, Tingwei Lu, Yongqin Zeng, Shaoxiong Zhan, Hai-Tao Zheng, and Hong-Gee Kim. Semanticache: Efficient kv cache compression via semantic chunking and clustered merging. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 19562–19566, 2026.

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.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations (ICLR)*, pp. 37228–37253, 2025.

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.

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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 31122–31137, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1584.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *International Conference on Machine Learning (ICML)*, pp. 58840–58850, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, et al. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026.

A Derivation of the V-Side Merging Error Bound

For a V-side fusion group $\mathcal{G}_i^V$, let V_i^{merge} denote its merged physical representation. With fixed attention weights $a_j \geq 0$, the original and merged contributions are $O_i^V = \sum_{j\in\mathcal{G}_i^V} a_j V_j$ and $\tilde{O}_i^V = \sum_{j\in\mathcal{G}_i^V} a_j V_i^{\text{merge}}$, respectively. The bound in Eq. 10 follows from the triangle inequality and nonnegative weights:

$$\begin{aligned}\left\|O_i^V - \tilde{O}_i^V\right\| &= \left\|\sum_{j\in\mathcal{G}_i^V} a_j V_j - \sum_{j\in\mathcal{G}_i^V} a_j V_i^{\text{merge}}\right\| \\ &= \left\|\sum_{j\in\mathcal{G}_i^V} a_j \left(V_j - V_i^{\text{merge}}\right)\right\| \\ &\leq \sum_{j\in\mathcal{G}_i^V} a_j \left\|V_j - V_i^{\text{merge}}\right\| \\ &\leq \left(\sum_{j\in\mathcal{G}_i^V} a_j\right) \max_{j\in\mathcal{G}_i^V} \left\|V_j - V_i^{\text{merge}}\right\|. \end{aligned} \tag{16}$$

The final inequality bounds each value deviation by the largest deviation within the fusion group. Thus, for fixed attention weights, the local error is bounded by the total attention weight of the group multiplied by its maximum deviation from the merged representation.

B Detailed Decoding Computation with Compact K/V States

This appendix substitutes the implicit state definitions in Eq. 14 into Eq. 15 to derive direct computation from compact physical states, without explicitly constructing $\widetilde{K}_{\text{ctx}}$ or $\widetilde{V}_{\text{ctx}}$. We use the same retained logical token set $\mathcal{C}_{\text{ctx}}$, newly generated token set $\mathcal{C}_{\text{gen}}$, and independent mappings M_K and M_V as in the main text. The reconstruction is approximate relative to the original uncompressed states; the following computation is an algebraic expansion of the attention expression for these approximate states.

B.1 K-Side Attention Logit Computation

For a logical token $j \in \mathcal{C}_{\text{ctx}}$, let its K-side physical token be $i = M_K(j)$ and its SimHash cluster be g. Substituting the K-state definition from Eq. 14 into the context logit in Eq. 15 yields three terms: the inner product over retained channels, mean compensation μ_i^K over pruned channels, and the additive logarithmic bias β_j:

$$\ell_j^{\text{ctx}} = \frac{q_{\text{new},\Omega_{\text{keep}}^{K,g}} K^T_{i,\Omega_{\text{keep}}^{K,g}} + \mu_i^K \sum_{c\in\Omega_{\text{prune}}^{K,g}} q_{\text{new},c}}{\sqrt{D}} + \beta_j, \tag{17}$$

where $\Omega_{\text{keep}}^{K,g}$ and $\Omega_{\text{prune}}^{K,g}$ are the retained and pruned channel sets of cluster g on the K side.

For newly generated tokens $t \in \mathcal{C}_{\text{gen}}$, since their corresponding keys K_t are not compressed yet, we use the standard inner product $\ell_t^{\text{gen}} = (q_{\text{new}} K_t^T)/\sqrt{D}$, then concatenate the compressed-context logits and generated-token logits:

$$\ell^{\text{ctx}} = [\ell_j^{\text{ctx}}]_{j\in\mathcal{C}_{\text{ctx}}}, \qquad \ell^{\text{gen}} = [\ell_t^{\text{gen}}]_{t\in\mathcal{C}_{\text{gen}}}, \tag{18}$$

$$a = \text{Softmax}\left([\ell^{\text{ctx}}, \ell^{\text{gen}}]\right), \tag{19}$$

where a is the attention weight vector, and each component is indexed by its corresponding context or generated token.

B.2 V-Side Attention Output Computation

During value aggregation, each logical token j is independently routed to the corresponding physical V token through $M_V(j)$. Expanding the V-state definition in Eq. 14, the retained-channel contribution is

$$O_{\text{keep}}^{\text{ctx}} = \sum_{j\in\mathcal{C}_{\text{ctx}}} a_j V_{M_V(j),\Omega_{\text{keep}}^V}, \tag{20}$$

where Ω^V_{keep} denotes a retained channel set on the V side.

For the pruned channels Ω^V_{prune}, the compensation vector $\mu^V_{\Omega^V_{\text{prune}}}$ is shared by all context logical tokens. Its weighted sum therefore equals this vector multiplied by the total context attention weight:

$$O^{\text{ctx}}_{\text{prune}} = \left(\sum_{j \in \mathcal{C}_{\text{ctx}}} a_j \right) \mu^V_{\Omega^V_{\text{prune}}}. \tag{21}$$

Then, the attention outputs of retained and compensated channels ($O^{\text{ctx}}_{\text{keep}}$ and $O^{\text{ctx}}_{\text{prune}}$) are restored by dimension-wise scatter to form O^{ctx}. The standard attention output of the newly generated sequence ($O^{\text{gen}} = \sum_{t \in \mathcal{C}_{\text{gen}}} a_t V_t$) is added to obtain the final attention output:

$$O_{\text{new}} = O^{\text{ctx}} + O^{\text{gen}}. \tag{22}$$

This output equals Eq. 15 for the implicitly reconstructed approximate states, without explicitly materializing the full context K/V matrices.

Note that when V-side channel fallback is not triggered, the compensation term $O^{\text{ctx}}_{\text{prune}}$ is not involved in computation, and V-side aggregation degenerates into V-side fusion, i.e., weighted aggregation over physical V tokens based on M_V.

C Hyperparameter Sensitivity Analysis

We first study the sensitivity of PACT-KV to merging thresholds on LLaMA3-8B-Instruct. Except for the investigated hyperparameter, all other ones follow the default configuration and use an equivalent budget of 256. We investigate the hyperparameters on Qasper and GovReport from LongBench, representing Single-Document QA and Summarization tasks, respectively.

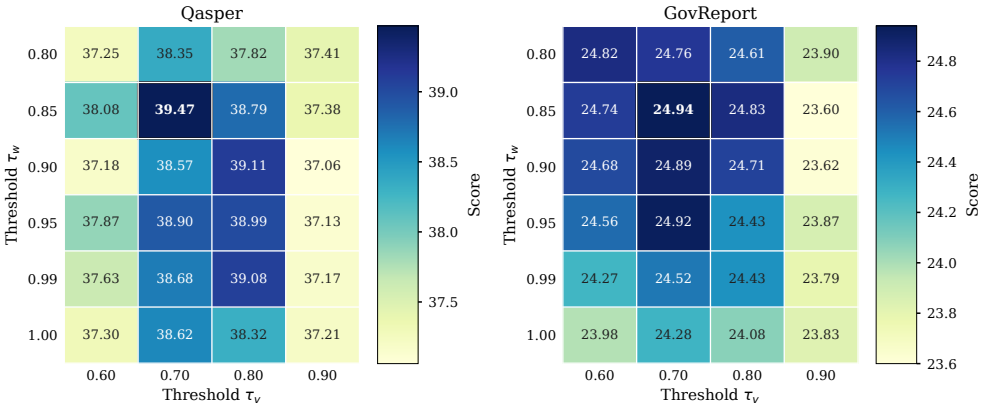


Figure 3: Hyperparameter sensitivity of τ_w and τ_v on Qasper and GovReport.

As shown in Fig. 3, both τ_w and τ_v have relatively stable effective ranges. When the thresholds are too low, tokens with different semantics or local attention roles may be incorrectly merged, damaging the original contextual information. When the thresholds are too high, fewer tokens can be merged, which reduces the compression benefit and also hurts performance. To avoid abrupt changes when τ_w is close to 1, we additionally test $\tau_w = 0.99$. It can be observed that high performance can be achieved when τ_w is from 0.85 to 0.95 and τ_v is 0.70.

We further study the SimHash bit count and K-channel pruning ratio on LLaMA3-8B-Instruct using Qasper and GovReport from LongBench. Except for the investigated hyperparameter, all other ones follow the default configuration and use an equivalent budget of 256.

Table 6: Sensitivity to the number of SimHash bits under a KV cache budget of 256.

Hash Bits C	Qasper	GovReport	Avg.
3	38.67	25.02	31.85
5	39.45	25.11	32.28
7	38.90	24.92	31.91
9	39.06	24.96	32.01
11	38.83	24.76	31.80

Table 7: Sensitivity to the K-side channel pruning ratio under a KV cache budget of 256.

ρ_K	Qasper	GovReport	Avg.
0.20	37.89	24.69	31.29
0.30	38.90	24.92	31.91
0.40	38.22	24.61	31.41
0.50	36.21	23.42	29.82
0.60	35.33	21.08	28.20

As shown in Tables 6 and 7, PACT-KV remains stable under different SimHash bit settings, indicating that the semantic clustering mechanism is not sensitive to hash granularity. For the K-channel pruning ratio, an overly high pruning ratio removes useful semantic information, while an overly low pruning ratio provides insufficient compression and leaves less budget for retaining tokens. Both cases therefore lead to performance degradation.

D LongBench Results under a Budget of 128

Tables 8 and 9 report the complete results under an equivalent KV cache budget of 128, using the same default hyperparameter setting as the main experiments. PACT-KV achieves average scores of 40.43, 46.07, and 42.67 on Llama-3-8B-Instruct, Llama-3.1-8B-Instruct, and Mistral-7B-Instruct, respectively, exceeding the reported compressed baselines for each model.

Table 8: LongBench performance under an equivalent KV cache budget of 128. Best results among compressed methods are highlighted in bold.

	Method	Single-Doc			Multi-Doc			Summ.			Few-shot			Synth.		Code		Avg.
		NQA	Qasp	MF	HQA	2Wiki	Musq	GovR	QM	MN	TREC	Triv	SAM	PCnt	PRe	Lcc	RB-P	
Llama-3-8B-Instruct	Full KV	22.43	43.85	44.16	46.70	36.78	21.13	30.13	22.60	27.78	75.00	90.48	42.41	5.50	64.50	59.23	54.26	42.93
	KV Cache Size = 128																	
	StreamingLLM	17.45	24.42	27.67	40.80	31.14	16.56	17.12	19.28	17.92	64.50	74.64	35.61	4.50	60.00	53.47	50.76	34.74
	H2O	**21.17**	32.19	32.50	44.23	30.43	18.75	21.94	20.96	24.54	70.00	86.25	35.94	5.00	57.50	55.00	52.47	38.05
	CaM	18.15	24.86	28.56	40.91	28.33	17.58	18.28	19.46	18.24	66.00	74.49	32.81	5.50	56.50	50.44	50.25	34.40
	SnapKV	21.07	31.44	**40.93**	44.65	34.29	20.70	20.41	19.58	23.02	73.00	72.31	39.36	4.00	**66.50**	56.19	48.42	38.49
	KeepKV	19.45	26.13	39.16	42.85	33.21	**22.09**	19.99	**21.40**	20.13	65.50	89.59	38.40	5.00	61.00	56.96	**54.30**	38.45
	PACT-KV	20.61	**33.98**	40.44	**45.59**	**35.95**	21.11	**23.25**	21.30	**24.99**	**73.50**	**89.91**	**40.12**	**7.50**	62.00	**57.03**	49.56	**40.43**
Llama-3.1-8B-Instruct	Full KV	28.18	45.53	54.85	56.52	46.66	31.20	35.18	25.36	27.21	74.00	91.65	43.82	8.38	99.50	65.16	58.60	49.49
	KV Cache Size = 128																	
	StreamingLLM	19.32	20.29	31.35	45.33	39.30	23.63	17.96	20.78	16.83	61.50	85.99	38.63	7.66	**99.50**	59.00	49.53	39.79
	H2O	22.57	27.27	36.53	50.69	42.82	27.92	26.22	22.37	**24.98**	69.00	89.19	37.53	7.34	**99.50**	58.16	49.55	43.23
	CaM	19.31	20.38	31.19	43.95	40.03	24.33	17.78	20.70	14.84	62.00	79.66	34.15	7.84	98.50	54.30	48.38	38.58
	SnapKV	23.36	29.73	47.92	53.52	45.06	28.41	21.79	22.99	21.82	**71.50**	88.98	40.42	**7.94**	97.50	59.15	52.11	44.51
	KeepKV	24.53	23.30	40.49	48.57	41.64	25.29	21.98	22.93	20.86	64.50	90.71	39.39	7.01	**99.50**	59.58	51.60	42.62
	PACT-KV	**26.88**	**31.46**	**49.16**	**55.63**	**45.37**	**29.32**	**27.05**	**23.75**	24.57	71.00	**91.39**	**41.62**	7.72	**99.50**	**60.10**	**52.57**	**46.07**
Mistral-7B-Instruct	Full KV	29.14	38.27	50.10	51.54	36.16	26.67	34.16	25.86	26.49	76.00	88.59	47.48	5.50	97.00	60.44	61.64	47.19
	KV Cache Size = 128																	
	StreamingLLM	19.42	19.06	26.95	38.12	27.22	16.04	17.10	19.45	16.05	62.50	86.14	39.93	**5.50**	51.00	53.58	50.63	34.29
	H2O	22.24	24.67	34.40	43.75	31.35	20.83	**28.18**	21.68	**25.84**	70.50	84.56	38.79	5.00	66.50	53.60	50.44	38.90
	CaM	11.75	17.21	27.26	27.67	**31.85**	13.75	15.92	18.74	17.62	64.00	80.35	32.18	3.35	3.50	43.99	47.41	28.53
	SnapKV	23.53	27.74	**43.77**	43.95	29.85	19.97	22.59	22.85	21.28	**71.00**	**89.40**	43.39	4.50	**93.50**	55.22	54.72	41.70
	KeepKV	22.42	24.66	35.80	38.57	29.47	18.41	22.10	23.19	20.93	63.00	88.00	43.71	3.50	89.00	**55.93**	**55.37**	39.63
	PACT-KV	**24.13**	**30.42**	43.13	**47.40**	31.37	**22.17**	26.27	**23.25**	24.38	68.50	89.36	**43.95**	5.00	93.00	55.70	54.72	**42.67**

Table 9: Comparison with SPARK- and ThinK-enhanced token-level baselines on LongBench category averages using Llama-3-8B-Instruct under an equivalent KV cache budget of 128.

Method	Single-Doc	Multi-Doc	Summ.	Few-shot	Synth.	Code	Avg.
SPARK+H2O	27.17	33.01	20.69	64.71	31.75	56.71	38.35
SPARK+SnapKV	**32.15**	33.20	21.34	62.90	34.38	52.07	38.85
ThinK+H2O	28.23	32.83	20.76	62.69	33.25	**57.60**	38.45
ThinK+SnapKV	30.87	33.61	19.89	64.58	33.50	56.46	39.17
PACT-KV	31.68	**34.22**	**23.18**	**67.84**	**34.75**	53.30	**40.43**