SELKV: SELECTIVE KV CACHE MERGING WITH PER-TOKEN MERGE-OR-DROP AND ATTENTION COMPENSATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) rely on a key-value (KV) cache whose memory footprint grows linearly with context length, creating a major inference bottleneck. Token merging methods reduce this cost by consolidating evicted tokens into retained cache entries rather than discarding them, but existing approaches merge every evicted token uniformly, regardless of whether its content is compatible with its merge target, and correct the resulting softmax under-weighting of merged positions, when they correct it at all, using raw merge counts that become unstable under aggressive compression. We propose SelKV, a training-free framework with two components: a soft cosine gate that modulates merge intensity per token based on value-vector similarity, continuously interpolating between full merge and drop, and an attention-ratio compensation mechanism that corrects softmax imbalance using a bounded, prefill-derived logit bias in place of unstable count-based correction. Across three models spanning MHA and GQA architectures on 16 LongBench datasets, paired bootstrap resampling shows the full method significantly recovers the loss otherwise incurred by blind merging ($p < 0.001$), with the compensation component alone contributing a further significant gain ($p = 0.009$), and SelKV substantially outperforms existing merging baselines (KVMerger, WeightedKV, and KeepKV's count-based compensation), with the margin widening further under aggressive compression. Against strong eviction baselines, SelKV matches performance overall while occasionally matching or slightly exceeding full-cache quality on distributed-evidence QA, and on the RULER benchmark it retains 95% of full-cache quality at 32k and 64k tokens. At 100k tokens, SelKV delivers a 3.3x decoding speedup over the full cache.

1 INTRODUCTION

Large language models (LLMs) achieve strong performance across diverse tasks, but their autoregressive inference is bottlenecked by the key-value (KV) cache, which stores past key and value vectors for attention computation. For a model with L layers and H attention heads of dimension d, the KV cache grows as $O(L \cdot H \cdot d \cdot n)$ with sequence length n, consuming multiple gigabytes even at moderate context lengths. As applications demand longer contexts, KV cache memory has become a primary constraint on batch size, throughput, and deployment cost.

Two families of methods have emerged for KV cache compression. *Eviction* methods (Zhang et al., 2023; Li et al., 2024b; Xiao et al., 2024; Cai et al., 2025) permanently discard low-importance tokens, reducing cache size but irreversibly losing the information they carry. *Merging* methods such as KVMerger (Wang et al., 2024), WeightedKV (Yuan et al., 2025), D2O (Wan et al., 2025), and LOOK-M (Wan et al., 2024) instead consolidate evicted tokens into retained cache entries, which can preserve more semantic content when information is distributed across many tokens, but which introduce two under-examined problems.

Problem 1: The uniform merge-or-drop decision. Most methods apply the same strategy to *all* evicted tokens: either merge all of them (KVMerger, WeightedKV, LOOK-M) or drop all of them (Zhang et al., 2023; Li et al., 2024b; Xiao et al., 2024). D2O (Wan et al., 2025) partitions tokens into drop and merge sets based on attention patterns, but this is a binary decision that still

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.

merges every token assigned to the merge set at full weight. No existing method modulates *how much* to merge on a continuous, per-token basis using representation similarity. EMS (Li et al., 2024a) similarly notes that "both over-merging and over-evicting yield sub-optimal performance," yet does not provide a per-token mechanism.

Problem 2: Softmax under-weighting of merged positions. When multiple tokens are merged into a single cache entry, that entry still receives roughly the same softmax attention as an unmerged token, despite representing several originals. KeepKV (Tian et al., 2024) formalizes this effect (Theorem 3.2): a position representing m merged tokens is systematically under-attended by a factor of $\sim m$, and proposes a logit-bias correction based on the raw merge count. Under heavy compression ($\geq$75% eviction), however, raw merge counts can reach $\sim$200 and can overweight tokens that carried little attention mass.

We address both problems with SelKV (Selective KV Cache Merging), a training-free framework with two complementary mechanisms that can be integrated into existing merging pipelines:

1. **Soft cosine gate** (Problem 1): For each evicted token, we compute the cosine similarity between its value vector and its merge target's value vector. The gate $g = \max(\cos_\mathrm{sim}, 0)$ then modulates merge intensity continuously: similar tokens merge fully ($g \approx 1$), orthogonal tokens are dropped ($g = 0$), and intermediate cases merge partially. No learned parameters or threshold tuning are required.
2. **Attention-ratio compensation** (Problem 2): We add a logit bias during decoding derived from prefill attention statistics, estimating how much attention mass each merged position should receive. Unlike naïve count-based compensation, this formulation is bounded by the redistributed attention mass and therefore remains stable under aggressive compression.

We evaluate on 16 LongBench datasets across three MHA/GQA models at 25% KV retention, and on RULER up to 64k tokens. Paired bootstrap tests validate the gate and compensation against controlled merging ablations; comparisons to merging baselines and eviction methods are reported in Section 4. Our contribution is a training-free mechanism that makes merging selective while retaining information that eviction discards.

2 Related Work

KV cache compression methods largely fall into three groups: token eviction, token merging, and quantization. Quantization methods such as KIVI (Liu et al., 2024) and ZipCache (He et al., 2024) reduce precision rather than token count and are largely complementary to our method, so we focus here on eviction and merging.

2.1 Token Eviction

Token eviction methods reduce the KV cache by permanently discarding entries deemed unimportant. Per-step methods such as H2O (Zhang et al., 2023) identify *Heavy Hitter* tokens that accumulate disproportionate attention mass and evict the rest at every decoding step, while preserving a recent-token window; VATP (Guo et al., 2024) extends this by observing that attention weight alone is an incomplete importance proxy, since a token's actual contribution is the product of its attention weight and its value magnitude. One-shot methods compress the cache once after prefill: StreamingLLM (Xiao et al., 2024) keeps only attention-sink tokens and a sliding recent window; SnapKV (Li et al., 2024b) uses an observation window over the last w query positions, smoothed by average pooling, to identify important positions per head; PyramidKV (Cai et al., 2025) applies a layer-adaptive budget. More recent one-shot methods include RocketKV (Behnam et al., 2025), EvolKV (Yu & Chai, 2025), and LAVa (Shen et al., 2025). All eviction methods permanently destroy the information carried by discarded tokens, which can be especially costly when evidence is spread across many positions, as in multi-document QA.

2.2 Token Merging

Rather than discarding evicted tokens, merging methods consolidate their information into retained cache entries. KVMerger (Wang et al., 2024) identifies sequences of tokens with similar key repre-

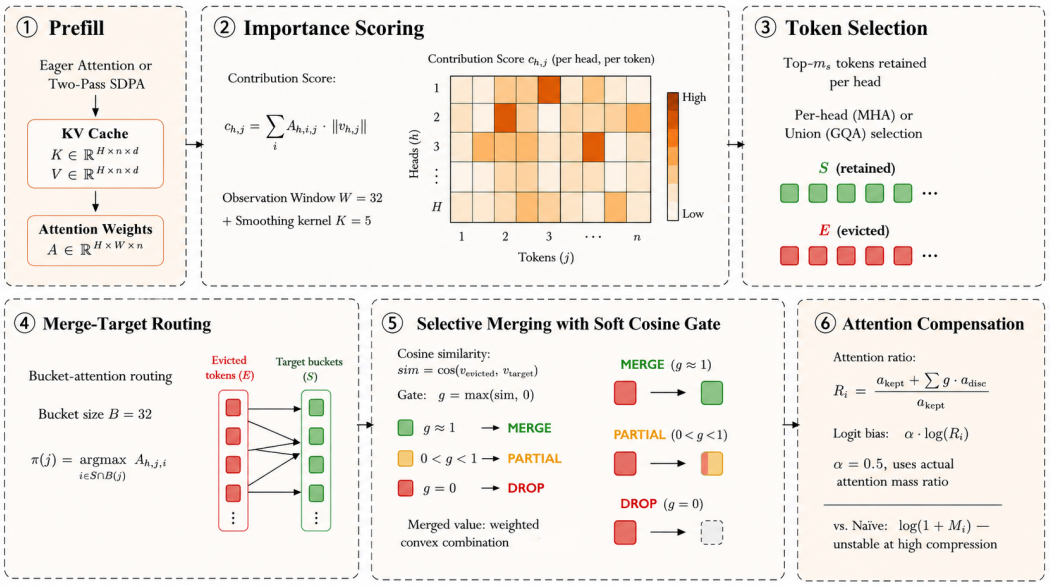

Figure 1: Overview of the selective KV cache compression pipeline. After prefill, the method scores token importance, selects retained tokens, routes evicted tokens to merge targets, applies the soft cosine gate, and compensates attention for merged positions before decode.

sentations and fuses them via a Gaussian kernel-weighted scheme. D2O (Wan et al., 2025) partitions tokens into drop and merge sets based on attention patterns but applies the same strategy to all tokens within each partition. LOOK-M (Wan et al., 2024) merges tokens within a local sliding window. WeightedKV (Yuan et al., 2025) applies attention-weighted convex combinations. ChunkKV (Liu et al., 2025) treats fixed-size semantic chunks as compression units. Most existing methods apply a **uniform** merge-or-drop decision: either *all* evicted tokens are merged at full weight, or *all* are dropped. EMS (Li et al., 2024a) explicitly observes that both over-merging and over-evicting yield sub-optimal performance, yet proposes no continuous per-token gating mechanism; our soft cosine gate fills this gap.

A further problem affects all merging methods: when multiple tokens are consolidated into one cache entry, that entry receives roughly the same softmax attention as a single unmerged token. KeepKV (Tian et al., 2024) formalizes this (Theorem 3.2) and proposes a logit bias based on raw merge counts, which we show in Section 4.4 is unstable at aggressive compression, unlike our bounded attention-ratio formulation. The majority of merging methods (KVMerger, WeightedKV, D2O, ChunkKV) do not incorporate any compensation at all.

3 METHOD

We present SelKV, a pluggable framework for selective KV cache compression. Given a pretrained transformer with L layers, H attention heads, and head dimension d, SelKV compresses the KV cache *once* after prefill and then decodes with the compressed cache. The framework introduces two training-free mechanisms that can be added to existing merging pipelines: (1) a soft cosine gate that decides per token how much to merge (§3.5), and (2) attention compensation that corrects the under-attention of merged positions (§3.6). Figure 1 illustrates the full pipeline: importance scoring (§3.2), token selection (§3.3), merge-target routing (§3.4), selective merging with soft cosine gate (§3.5), and attention compensation (§3.6). Appendix A lists the exact configuration used for every model $\times$ context-length combination in our experiments.

3.1 Observation Window and Cache Setup

Let $\mathbf{x} = (x_1, \ldots, x_n)$ denote the input sequence of length n. We need (i) the full KV cache $\{(\mathbf{K}^{(\ell)}, \mathbf{V}^{(\ell)})\}_{\ell=1}^{L}$, where $\mathbf{K}^{(\ell)}, \mathbf{V}^{(\ell)} \in \mathbb{R}^{H \times n \times d}$; and (ii) attention weights over an observation window of W query positions for importance scoring and routing. A naïve eager prefill would materialize $O(n^2)$ attention, infeasible at long contexts. Instead, we use a *two-pass* strategy: (1) run SDPA prefill to build the full KV cache without materializing attention matrices, then (2) re-run only the last W tokens with eager attention against the cached keys, producing attention matrices of size $H \times W \times n$ per layer. This reduces prefill memory from $O(n^2)$ to $O(W \cdot n)$ while yielding the same attention weights used by SnapKV-style observation windows.

Given a target retention ratio $\rho \in (0, 1)$, we set the target cache length to $m = \lfloor \rho \cdot n \rfloor$. A *recent window* of the last $m_r = 16$ tokens is always retained (Xiao et al., 2024). The remaining $m_s = m - m_r$ slots are filled by importance-based selection. After compression, we switch to SDPA for decode.

3.2 Importance Scoring

Prior methods score token importance by cumulative attention alone (Zhang et al., 2023). Following VATP (Guo et al., 2024), we observe that attention weight alone is an incomplete proxy: a token may receive high attention but carry a near-zero value vector, contributing little to the output. We define the *contribution score* of token j at layer ℓ and head h as:

$$c_{h,j}^{(\ell)} = \sum_{i \in \mathcal{W}} A_{h,i,j}^{(\ell)} \cdot \|\mathbf{v}_{h,j}^{(\ell)}\|_2 \tag{1}$$

where $\mathcal{W}$ denotes the query positions used for scoring. Following SnapKV (Li et al., 2024b), we restrict $\mathcal{W}$ to the last $W = 32$ positions and apply average-pooling smoothing with kernel size $K = 5$.

3.3 Token Selection

Different layers exhibit distinct attention patterns (Cai et al., 2025), so we perform token selection independently at each layer.

Per-head selection (MHA models). Each head h at layer ℓ independently selects its top-m_s tokens by contribution score:

$$\mathcal{S}_h^{(\ell)} = \mathrm{top}_{m_s}\big(\{c_{h,j}^{(\ell)}\}_{j=1}^{n-m_r}\big) \cup \{n - m_r + 1, \ldots, n\} \tag{2}$$

The complementary set $\mathcal{E}_h^{(\ell)} = \{1, \ldots, n\} \setminus \mathcal{S}_h^{(\ell)}$ contains the evicted tokens for head h.

Union selection (GQA models). On GQA models, each KV head serves G query heads. We average per-query-head importance scores across the G heads sharing a KV head, then each KV head independently selects its top-m_s tokens; we take the *set union* across heads, trimming by mean aggregate score if the union exceeds the layer budget m. Merging requires a shared token set across KV heads, since the merge operation (Eq. 5) combines value vectors at shared positions; per-head gather, where different heads keep different token sets, is structurally incompatible with merging because a candidate merge target might exist for one head but not another. We verify in Appendix H that this choice is immaterial to SelKV's gains, and we apply the same union selection policy to the eviction baselines on GQA models (Section 4.1) so that the selection rule is held fixed across methods and cannot by itself explain any measured gain. Thus, the selection rule is held fixed across methods; the ablation isolates the gate and compensation.

3.4 Merge-Target Routing

Each evicted token $j \in \mathcal{E}_h^{(\ell)}$ is assigned a merge target $\pi_h(j) \in \mathcal{S}_h^{(\ell)}$ via *bucket-attention routing*: the sequence is partitioned into positional buckets of size $B = 32$, and each evicted token is routed to the kept token within the same bucket that received the highest attention from it during prefill:

$$\pi_h(j) = \operatorname*{argmax}_{i \in \mathcal{S}_h^{(\ell)} \cap \mathcal{B}(j)} A_{h,j,i}^{(\ell)} \tag{3}$$

This reduces routing complexity to $O(H \cdot |\mathcal{E}| \cdot B)$ from $O(H \cdot |\mathcal{S}| \cdot |\mathcal{E}| \cdot d)$ for full cosine routing. We sweep B in Appendix G and use $B = 32$ by default, which balances positional locality against global routing quality.

3.5 Selective Merging with Soft Cosine Gate

Uniformly merging all evicted tokens is harmful when an evicted token's value vector is dissimilar to its merge target's value vector. For each evicted token j routed to target $\pi_h(j)$ at head h, we compute the cosine similarity between their value vectors:

$$\text{sim}_{h,j} = \frac{\mathbf{v}_{h,j}^{(\ell)} \cdot \mathbf{v}_{h,\pi_h(j)}^{(\ell)}}{\|\mathbf{v}_{h,j}^{(\ell)}\|_2 \cdot \|\mathbf{v}_{h,\pi_h(j)}^{(\ell)}\|_2} \tag{4}$$

The soft gate is $g_{h,j} = \max(\text{sim}_{h,j},\ 0)$: $g = 0$ (drop) for dissimilar vectors, $0 < g < 1$ (partial merge) for moderate similarity, $g \approx 1$ (full merge) for high similarity.

Gate signal. Because the gate modulates value averaging, we use value cosine to measure directional compatibility. L2 distance is scale-sensitive, dot product conflates magnitude with direction, and attention score measures query-key relevance rather than value compatibility; routing already uses attention for target selection. A controlled gate-space ablation validates this choice (Section 4.3, Appendix D).

The merged value for a kept token i is:

$$\tilde{\mathbf{v}}_{h,i}^{(\ell)} = \frac{a_{h,i} \cdot \mathbf{v}_{h,i} \ + \ \sum_{j:\pi_h(j)=i} g_{h,j} \cdot a_{h,j} \cdot \mathbf{v}_{h,j}}{a_{h,i} \ + \ \sum_{j:\pi_h(j)=i} g_{h,j} \cdot a_{h,j}} \tag{5}$$

where $a_{h,i} = \sum_{q \in \mathcal{W}} A_{h,q,i}$ is the cumulative attention over the observation window. When $g = 0$ for all evicted tokens, Eq. 5 reduces to pure drop; when $g = 1$, it reduces to standard attention-weighted merging.

Key handling. Evicted keys are discarded; only retained keys remain, while their values are updated by Eq. 5. We do not modify token positions or apply RoPE re-indexing; an early repositioning variant was dropped after harming retrieval (Appendix B).

3.6 Attention Compensation

As established by KeepKV (Tian et al., 2024) (Theorem 3.2), softmax attention systematically underweights merged positions. KeepKV compensates by adding $\log(1 + M_i)$ to attention logits, where M_i is the raw merge count. Under heavy compression ($\geq$75% eviction), merge counts can reach $\sim$200, and we find this count-based correction is not merely less stable but actively harmful (Section 4.4, Appendix E). We replace raw counts with an attention-ratio derived from the prefill pass:

$$R_{h,i} = \frac{a_{h,i}^{\text{kept}} + \sum_{j \to i} g_{h,j} \cdot a_{h,j}^{\text{disc}}}{a_{h,i}^{\text{kept}}} \tag{6}$$

On GQA models, we average $R_{h,i}$ across KV heads and add $\alpha \cdot \log(R_i)$ (with $\alpha = 0.5$) to the attention logits during decoding; on MHA models with per-head selection, the bias is applied per head without cross-head averaging. Unlike raw merge counts, R_i is bounded by the ratio of retained-to-total attention mass, so positions that absorbed negligible attention receive little or no boost, weighting the correction by actual information content rather than token count.

We fix $\alpha = 0.5$ throughout; a post-hoc sweep over $[0, 1]$ on both GQA models shows only modest variation (Appendix F).

4 Experiments

4.1 Setup

Models. We evaluate on three models spanning two attention architectures: LongChat-7B-v1.5-32k (MHA, 32 KV heads), LLaMA-3.1-8B-Instruct (GQA, 8 KV heads), and Gemma-2-9B-IT

(GQA, 8 KV heads, alternating sliding-window attention). LongChat uses 31,500-token inputs; the remaining models use 3,500-token inputs, near Gemma-2's 8,192-token context ceiling. All experiments use FP16 inference on NVIDIA H100 80GB GPUs.

Benchmark. We evaluate on the 16 English datasets of LongBench (Bai et al., 2024) (6 task categories) with standard per-dataset metrics, 200 samples per task (500 for code tasks), and separately on RULER (Hsieh et al., 2024) (11 harder retrieval tasks) at 4k–64k tokens to test whether quality holds at contexts beyond LongBench's evaluated range.

Baselines. We compare against one-shot eviction methods SnapKV (Li et al., 2024b), LOOK-M (Wan et al., 2024) (merging), and PyramidKV (Cai et al., 2025), and, to directly validate our two contributions against the methods they are positioned against, we additionally benchmark KVMerger (Wang et al., 2024), WeightedKV (Yuan et al., 2025), and KeepKV's count-based compensation (Tian et al., 2024) (Section 4.4). On GQA models, every baseline uses the same union token-selection rule as SelKV (Section 3.3), so differences in Table 1 and Table 3 isolate the gate and compensation rather than the selection policy. D2O's official implementation supports only LLaMA-2, precluding evaluation on our GQA models; conceptually, D2O applies uniform merging within its merge group, exactly the strategy our gate replaces with per-token modulation. We similarly did not extend the KeepKV and WeightedKV comparisons to Gemma-2 within this study's timeframe (Table 3); both are evaluated on LLaMA-3.1-8B, where the same qualitative pattern holds, and completing the Gemma-2 columns is left to future work rather than reflecting an architectural incompatibility.

Unless noted, we use contribution-based importance scoring, observation window $W = 32$, smoothing kernel $K = 5$, bucket size $B = 32$, recent window $m_r = 16$, and compensation factor $\alpha = 0.5$ for every model and context length, with the exception of the selection rule (per-head on the MHA model, union on the two GQA models; Section 3.3). Appendix A lists the exact configuration for every model $\times$ context-length combination evaluated in this paper.

4.2 Cross-Model Results

Table 1 reports per-dataset results across three models against the eviction and single-merging baselines. SelKV gives the top compressed result on both GQA models: Gemma-2 (42.31, -0.67) and LLaMA-3.1 (39.81, -0.73). On the MHA model LongChat (31.5k tokens), PyramidKV leads. To control for context length rather than architecture, we also evaluate LLaMA-3.1 at 31.5k tokens (Appendix L): SelKV and SnapKV tie exactly (48.46), both ahead of PyramidKV and LOOK-M, indicating that SelKV's edge on GQA at short context is a real architecture effect and not purely a short-context artifact, though it does shrink to parity with the strongest eviction baseline at longer context. At the category level on Gemma-2, SelKV leads on four of six categories (Single-Doc QA, Multi-Doc QA, Summarization, Few-shot), while SnapKV leads only on Code, where exact token identity matters (Appendix J).

Table 1: LongBench results (16 datasets) at 25% KV retention across three models. Δ: gap to Full Cache. Best compressed method per task in **bold**.

		Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		Overall		
LLM	**Method**	NrtQA	Qasp	MF-en	HpQA	2Wiki	Musi	GovR	QMSm	MNws	TREC	TrvQA	SAMs	PCnt	PRet	LCC	RB-P	**Avg**	Δ	**Avg Rank**
Gemma-2	Full Cache	22.93	40.79	46.68	47.25	48.74	22.84	27.63	19.47	24.81	67.50	91.92	43.56	7.00	33.50	72.78	70.33	42.98	–	–
	SnapKV	22.22	40.01	**46.78**	46.90	46.54	22.00	24.63	19.28	22.30	66.50	91.92	42.62	7.00	33.50	**72.62**	69.75	42.16	−0.82	2.06
	LOOK-M	22.66	36.75	45.93	**46.97**	46.05	22.34	23.98	19.18	22.73	55.50	**92.17**	42.25	7.00	33.50	72.07	**69.84**	41.18	−1.80	2.56
	PyramidKV	22.43	39.70	46.71	46.37	**47.30**	22.41	24.46	18.94	22.12	66.50	91.92	42.36	7.00	33.50	72.53	69.10	42.08	−0.90	2.50
	SelKV	**23.32**	**40.01**	46.44	46.67	46.84	**22.90**	**24.94**	**19.33**	**22.79**	**67.00**	91.76	**42.89**	**7.00**	**33.50**	72.24	69.35	**42.31**	**−0.67**	**1.69**
LLaMA-3.1	Full Cache	21.63	40.24	49.54	39.04	35.23	18.84	32.37	21.00	27.15	70.00	89.00	43.68	3.58	34.00	64.87	58.41	40.54	–	–
	SnapKV	21.33	**39.00**	49.71	39.25	35.86	18.88	27.97	21.04	23.59	**67.50**	89.63	42.00	**4.08**	34.00	**65.02**	**57.73**	39.79	−0.75	2.06
	LOOK-M	20.84	33.18	44.92	38.96	35.36	17.01	**28.33**	**21.59**	**24.84**	65.00	89.35	41.97	4.00	**35.00**	63.80	56.31	38.78	−1.76	3.00
	PyramidKV	21.60	38.04	49.20	**40.04**	34.77	18.65	27.63	20.76	23.34	67.00	89.77	**42.37**	**4.08**	34.00	63.83	57.19	39.52	−1.02	2.63
	SelKV	**21.60**	38.25	**50.01**	39.44	**35.91**	**19.55**	27.94	21.37	23.63	67.00	**89.95**	42.06	4.03	34.50	64.34	57.31	**39.81**	**−0.73**	**1.75**
LongChat	Full Cache	21.01	29.31	43.37	33.08	23.44	14.15	30.91	22.67	26.60	66.50	84.09	22.16	0.00	30.50	50.06	58.17	34.75	–	–
	SnapKV	**20.71**	27.33	39.42	**34.73**	22.66	**14.18**	26.52	22.54	13.25	64.00	**83.78**	27.68	0.00	29.00	43.35	**57.96**	32.94	−1.81	2.00
	LOOK-M	19.86	23.16	29.05	31.78	21.58	11.93	22.55	22.39	15.24	55.50	74.33	25.93	0.00	23.25	47.76	56.86	30.07	−4.68	3.38
	PyramidKV	20.17	**28.00**	**41.68**	34.04	**22.76**	13.97	**27.29**	**23.07**	**22.60**	**64.50**	82.34	23.79	0.00	30.50	**50.87**	57.55	**33.95**	**−0.80**	**1.63**
	SelKV	20.48	27.77	33.69	31.02	20.96	13.11	25.66	22.24	18.66	61.00	78.04	**29.54**	**0.00**	**38.50**	48.50	56.18	32.83	−1.92	2.63

Table 2: Staged ablation on LLaMA-3.1-8B (16 datasets, 25% KV retention), with paired bootstrap 95% confidence intervals (10,000 iterations) against the row above. "Evict only" uses our token selection but drops evicted tokens without merging.

Configuration	Gate	Comp	Merge	Avg (16 ds)	Δ vs. merge-all (95% CI, p)
Full Cache	–	–	–	40.54	–
Eviction only	–	–	–	39.69	$+0.69$ $[+0.35, +1.03]$, $p < 0.001$
Merge all (no gate)	–	–	✓	39.01	– (reference)
gate only	✓	–	✓	39.57	–
gate + comp(ours)	✓	✓	✓	**39.81**	$+0.80$ $[+0.47, +1.14]$, $p < 0.001$

Multi-document QA is where the structural advantage of merging over eviction is clearest: on several settings the compressed cache exceeds full-cache scores (e.g., LLaMA-3.1 HotpotQA 39.44 vs. 39.04, MuSiQue 19.55 vs. 18.84; Gemma-2 MuSiQue 22.90 vs. 22.84). These deltas are individually small; we treat them as suggestive rather than conclusive and discuss their statistical status in Section 4.3.

4.3 Statistical Validation of the Central Claim

Because average deltas of a few tenths of a point over 16 datasets cannot be distinguished from noise without repeated-run statistics, we ran paired bootstrap resampling (10,000 iterations over per-example scores) on a controlled ablation where every configuration shares the same token selection, hyperparameters, and code path, varying only the gate and compensation. Table 2 reports the staged ablation with bootstrap confidence intervals.

Two results anchor our claim. First, blind merging is a real and significant failure mode: eviction beats merge-all by $+0.69$ points ($p < 0.001$), and gate+compensation recovers essentially all of this loss ($+0.80$ vs. merge-all, $p < 0.001$), with the compensation component alone contributing a further significant gain over the gate-only configuration ($+0.24$, $p = 0.009$). Second, the aggregate delta between the full method and eviction-only (39.81 vs. 39.69, $+0.12$) has a 95% CI of $[-0.16, +0.38]$ and does not reach significance ($p = 0.20$), and the long-context control (Appendix L) ties SnapKV exactly. We do not claim SelKV outperforms strong eviction; we claim, with statistical support, that SelKV eliminates the loss otherwise incurred by merging and reaches parity with eviction while retaining merging's advantage on distributed-evidence tasks.

A controlled gate-space ablation isolating the gate signal alone (token selection, routing, and code path held fixed; full per-task table in Appendix D) confirms the value-space design of Section 3.5: no gate scores 38.96, an attention-based gate 39.42, a key-cosine gate 39.37, and our value-cosine gate 39.59. The gates are close on aggregate, but value cosine improves content-critical tasks; full per-task results are in Appendix D.

4.4 Comparison Against Merging Baselines

The comparison that actually validates our two contributions is against the methods they are positioned to improve on: count-based compensation (KeepKV) and uniform merging (KVMerger, WeightedKV). Table 3 reports this comparison at 25% retention.

SelKV outperforms KeepKV's count-based compensation by 0.91 points on the same model and evaluation, KVMerger by 1.54/1.61 points, and WeightedKV by 1.29 points (all $p < 0.001$ by paired bootstrap), with the largest gaps on information-retrieval tasks where indiscriminate merging corrupts value representations (Qasper $+4.89$/$+4.78$ vs. KVMerger, MultifieldQA-en $+4.31$/$+4.96$, TREC $+6.0$/$+4.5$). The cosine gate also transfers as a plug-in: adding it to LOOK-M improves Gemma-2 from 41.18 to 41.35 and LLaMA-3.1 from 38.78 to 38.86.

The advantage widens under aggressive compression. At a 10% retention budget, where each merge decision carries the most weight because few merge targets remain, the gap between SelKV and other merging methods grows substantially (LLaMA-3.1-8B, 16 datasets, full figures in Appendix K): SelKV drops only 1.75 points from its 25% score (39.81 $\rightarrow$ 38.06), while KVMerger drops 5.29 (38.27 $\rightarrow$ 32.98) and WeightedKV drops 4.81 (38.52 $\rightarrow$ 33.71).

Table 3: SelKV against merging baselines at 25% KV retention (16 datasets). KeepKV row uses our token selection and gate, varying only the compensation term (count-based vs. our attention-ratio), isolating the compensation contribution. KeepKV and WeightedKV are not yet evaluated on Gemma-2 ("–"); see Section 4.1 for why.

Method	LLaMA-3.1-8B	Gemma-2-9B
Full Cache	40.54	42.98
SelKV (ours)	**39.81**	**42.31**
SnapKV (eviction)	39.79	42.16
KeepKV (count-based compensation)	38.90	–
KVMerger	38.27	40.70
WeightedKV	38.52	–

4.5 Long-Context Quality (RULER)

LongBench quality was validated at 3.5k tokens for two of our three models; to test whether this generalizes, we run RULER (Hsieh et al., 2024) (11 harder retrieval tasks: single-needle, multi-key, multi-query, multi-value, common/frequent word extraction, variable tracking) on LLaMA-3.1-8B-Instruct at 4k, 32k, and 64k tokens (Table 4). Gemma-2-9B-IT's 8,192-token context ceiling precludes evaluation at 32k/64k on that model, so long-context confirmation is on LLaMA-3.1 only.

Table 4: RULER (11 tasks) on LLaMA-3.1-8B-Instruct at 25% KV retention. We report Full Cache and SelKV only; an eviction baseline (e.g. SnapKV) on RULER is left to future work (see text).

Method	4k	32k	64k
Full Cache	92.4	84.4	81.2
SelKV (25%)	80.5	80.5	77.0

RULER has no eviction baseline; a 31.5k-token LongBench control (Appendix L) shows SelKV ties SnapKV (48.46).

SelKV retains 95% of full-cache quality at both 32k and 64k tokens, indicating that quality does not collapse as context grows well beyond the 3.5k–31.5k range validated on LongBench. The lower 4k retention is consistent with the smaller absolute cache budget at a fixed ratio.

4.6 Latency Analysis

Figure 2 shows wall-clock time on LLaMA-3.1-8B across prompt lengths from 512 to 100k tokens. At short contexts ($\leq$8k), decode throughput matches the full cache ($\sim$50 tok/s). Beyond 16k tokens, the smaller KV cache decodes faster: at 100k tokens we observe 28 tok/s vs. 8.4 tok/s for the full cache (3.3$\times$). Compression overhead stays below 5% of total time at 8k+. We note that this benchmark measures throughput scaling only; generation quality is validated up to 64k tokens (RULER) and 31.5k tokens (LongBench), not at 100k.

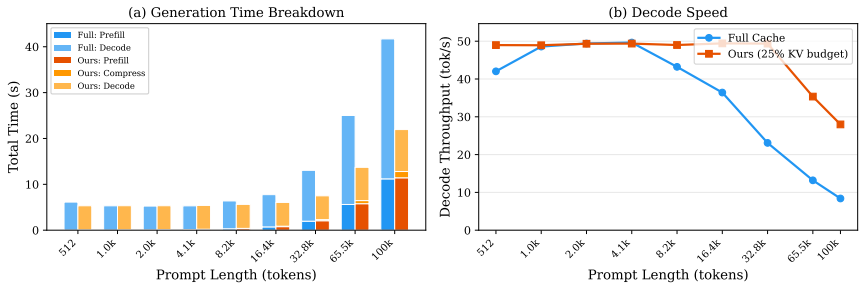

Figure 2: Latency on LLaMA-3.1-8B (256 generated tokens, H100, 25% KV retention budget). (a) Total time breakdown. (b) Decode throughput.

5 LIMITATIONS

- **SelKV reaches parity with, not superiority over, strong eviction** overall (Section 4.3). At aggressive (10%) retention, eviction's absolute lead over merging widens, even as SelKV's margin over other merging methods widens by more.
- **Merging is a weaker fit for MHA and code tasks:** PyramidKV leads on LongChat (MHA), and code completion favors SnapKV, where exact token identity matters.
- **Evaluation covers three models and short generations** (at most 500 decoded tokens); broader model coverage and direct measurement of compensation drift over longer generations are left to future work. RULER (Section 4.5) is reported against Full Cache only, without an eviction baseline at 32k–64k.
- **The 100k-token speedup is a throughput measurement;** generation quality is validated up to 64k tokens (RULER).

6 CONCLUSION

We present SelKV, a training-free KV-cache compression framework combining a soft cosine gate and attention-ratio compensation. SelKV eliminates the significant loss incurred by blind token merging, reaches statistical parity with strong eviction baselines, and substantially outperforms existing merging methods, a margin that widens under aggressive compression. We do not claim merging outperforms eviction; we show, with bootstrap-validated statistics, that per-token merge quality is the design axis that determines whether merging is viable at all as a compression strategy. The result is a focused, honestly-scoped contribution: a simple mechanism that converts merging from a strategy that sometimes helps and sometimes corrupts the cache into one that is safe by construction, with a clear account of where its advantage lies and where it does not.

AI USE STATEMENT

Generative AI tools were used to assist with editing prose and LaTeX formatting during the preparation of this manuscript. All experimental design, code, results, and statistical analyses are the authors' own work. All AI-assisted text was reviewed and edited by the authors, who take full responsibility for the final content, including all claims and reported numbers.

ETHICS STATEMENT

This work uses publicly available pretrained language models and the public LongBench and RULER benchmarks. It does not involve human subjects, crowdsourcing, or private data. As with any inference-efficiency method, faster and cheaper long-context inference could in principle lower the cost of deploying LLMs for both beneficial and harmful applications; we are not aware of concerns specific to KV cache compression beyond those general to LLM inference efficiency.

REPRODUCIBILITY STATEMENT

The method is fully specified in Section 3 (Equations 1–6), with hyperparameters listed in 4.1 and Appendix M. All datasets (LongBench, RULER) are public. Our statistical methodology (paired bootstrap resampling, 10,000 iterations, seed 42) is described in Section 4.3 and Appendix M. Anonymized code is included in the supplementary material.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In *International Conference on Machine Learning (ICML)*, 2025.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, et al. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Conference on Language Modeling (COLM)*, 2025.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention score is not all you need for token importance indicator in KV cache reduction: Value also matters. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

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)*, 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 *First Conference on Language Modeling (COLM)*, 2024.

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. *arXiv preprint arXiv: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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024b.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. *arXiv preprint arXiv:2502.00299*, 2025.

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)*, 2024.

Yiqun Shen, Song Yuan, Zhengze Zhang, Xiaoliang Wang, Daxin Jiang, and Cam-Tu Nguyen. LAVa: Layer-wise KV cache eviction with dynamic budget allocation. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 13672–13692, 2025.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. KeepKV: Eliminating output perturbation in KV cache compression for efficient LLMs inference. *arXiv preprint arXiv:2504.09936*, 2024.

Zhongwei Wan, Ziang Wu, Che Liu, Jinfa Huang, Zhihong Zhu, Peng Jin, Longyue Wang, and Li Yuan. LOOK-M: Look-once optimization in KV cache for efficient multimodal long-context inference. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 4065–4078, 2024.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, et al. D2O: Dynamic discriminative operations for efficient long-context inference of large language models. In *International Conference on Learning Representations (ICLR)*, 2025.

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.

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.

Bohan Yu and Yekun Chai. EvolKV: Evolutionary KV cache compression for LLM inference. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 1673–1689, 2025.

Jian Yuan, Ziwei He, Haoli Bai, Jingwen Leng, and Bo Jiang. WeightedKV: Attention scores weighted key-value cache merging for large language models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, 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)*, 2023.

A Per-Experiment Configuration

Table 5 lists the exact configuration used for every model $\times$ context-length combination reported in this paper. All settings not listed follow the defaults in Section 4.1.

Table 5: Per-experiment configuration. RoPE repositioning of merge-target keys is disabled in every row; see Appendix B.

Model	**Context**	**Benchmark**	**Selection rule**	B	α
LongChat-7B-v1.5-32k	31.5k tokens	LongBench	Per-head (MHA)	32	0.5
LLaMA-3.1-8B-Instruct	3.5k tokens	LongBench	Union (GQA)	32	0.5
LLaMA-3.1-8B-Instruct	31.5k tokens	LongBench	Union (GQA)	32	0.5
LLaMA-3.1-8B-Instruct	4k–64k tokens	RULER	Union (GQA)	32	0.5
Gemma-2-9B-IT	3.5k tokens	LongBench	Union (GQA)	32	0.5

B RoPE Repositioning: A Negative Finding

An early variant of SelKV re-indexed the positional embedding of each merge target after compression, on the hypothesis that a merged entry's position should reflect the span of tokens it summarizes rather than its original position. We tested this on LLaMA-2-7B and found it harms retrieval substantially (PassageRetrieval-en 5.0 with repositioning vs. 38.5 without) and, at longer contexts, induces repetition loops during decoding. We attribute this to a mismatch between the repositioned key and the RoPE-encoded query used to retrieve it: routing (Eq. 3) and the gate (Section 3.5) both assume the merge target's key retains its original positional encoding, and repositioning breaks this assumption. RoPE repositioning is therefore disabled in every configuration reported in this paper (Table 5) and is not part of the method described in Section 3.

C Computational Complexity

The dominant prefill cost is the SDPA forward pass, $O(L \cdot H \cdot n^2 \cdot d)$, identical to any method that builds a KV cache. The two-pass eager re-pass over the observation window adds $O(L \cdot H \cdot W \cdot n \cdot d)$, linear in n for fixed W. The compression step itself is efficient:

- **Importance scoring**: $O(L \cdot H \cdot n \cdot (W + d))$.
- **Selection**: $O(L \cdot H_{\text{kv}} \cdot n \log m)$ per-KV-head top-k, followed by set union.
- **Routing**: $O(L \cdot H \cdot |\mathcal{E}| \cdot B)$.
- **Soft gate**: $O(L \cdot H \cdot |\mathcal{E}| \cdot d)$, one cosine similarity per evicted token.
- **Merge**: $O(L \cdot H \cdot |\mathcal{E}| \cdot d)$, vectorized scatter-add.
- **Attention compensation** (decode): $O(m)$ per step.

Total compression overhead is 0.04s for a 2048-token sequence on an H100 GPU, amortized over the entire decode phase.

D GATE-SPACE ABLATION: FULL PER-TASK RESULTS

Table 6 reports the full 16-task version of the controlled per-task comparison summarized in Section 4.3 (LLaMA-3.1-8B, 16 datasets, 25% retention, gate only, no compensation; hyperparameters, token selection, and code path identical across columns).

Table 6: Per-task gate-space ablation, sorted by spread (largest gate effect first).

Task	No gate	Attention gate	Key cosine	Value cosine (ours)
TREC	62.50	66.00	64.50	**66.50**
PassageCount	3.50	1.53	3.53	**3.53**
TriviaQA	88.16	**89.32**	90.16	89.61
RepoBench-P	54.84	55.62	56.11	**56.63**
SAMSum	40.96	**42.01**	42.03	41.86
Qasper	37.96	38.90	37.88	**38.36**
GovReport	26.78	**27.65**	26.99	27.62
2WikiMQA	35.17	35.62	**35.96**	35.95
PassageRetrieval	33.50	33.50	**34.00**	**34.00**
NarrativeQA	21.98	**22.08**	21.65	21.60
HotpotQA	**40.28**	40.24	39.91	39.83
MultifieldQA-en	**50.41**	50.47	50.06	50.09
MultiNews	23.34	**23.62**	23.22	23.54
QMSum	**21.11**	21.05	20.82	21.02
LCC	63.81	63.94	63.78	**64.04**
MuSiQue	19.01	**19.23**	19.25	19.24
Average	38.96	39.42	39.37	**39.59**

E MERGING-BASELINE DETAILS

KeepKV count-based compensation, full sweep. On LLaMA-2-7B (16 datasets): no compensation scores 32.05; count-based compensation at its default strength scores 31.24 (-0.81); per-layer sweeps across correction strengths all score worse than no compensation (30.98–31.30). We also confirmed the same qualitative pattern on LLaMA-3.1-8B (38.90 vs. SelKV's 39.81, Table 3), so the failure of count-based compensation is not specific to the older LLaMA-2 architecture.

F ATTENTION-RATIO SENSITIVITY (α)

Table 7 reports the full sweep across $\alpha \in [0, 1]$ referenced in Section 3.6, on both GQA models (16 datasets, 25% retention), expressed as delta from the default $\alpha = 0.5$.

Table 7: α sensitivity (delta from default $\alpha = 0.5$), both main GQA models, 16 datasets, 25% retention. Improvement is monotonic on both models and the full 0–1 range spans at most 0.31 points.

α	**LLaMA-3.1** (Δ)	**Gemma-2** (Δ)
0.00 (gate only)	-0.17	-0.17
0.25	-0.13	-0.05
0.50 (default)	0	0
0.75	$+0.03$	$+0.12$
1.00	$+0.08$	$+0.14$
Range	0.25	0.31

The curve is monotonic and flattens well before $\alpha = 1$: on LLaMA-3.1, three quarters of the total 0–1 range's movement (0.25 of 0.25 points) is already realized by $\alpha = 0.75$, and Gemma-2 shows the same pattern (0.12 of 0.14 points). We report $\alpha = 0.5$ as the default throughout this paper, chosen prior to running this sweep; the sweep was conducted afterward, specifically to check robustness rather than to select the reported value, and its flatness is the evidence that the attention-ratio formulation does not require careful tuning, in contrast to raw merge counts (Appendix E).

G Bucket Size Sensitivity

On LLaMA-3.1-8B (16 datasets, 25% retention): $B = 16$ scores 39.01, $B = 32$ (default) scores 39.81, and $B = 64$ scores 38.88. $B = 16$ over-constrains available merge targets within each bucket; $B = 64$ dilutes the positional locality that makes bucket-attention routing effective.

H Union vs. Mean-Reduction Selection Fairness Ablation

On LLaMA-3.1-8B-Instruct (16 datasets), union selection (a token is kept if any KV head selects it) and mean-reduction selection (average importance across heads, global top-k) differ by only 0.07 points, with 82–96% token overlap (Jaccard index) across layers. Applying the same union-vs-mean comparison to SnapKV itself yields a 0.03-point difference. This confirms that SelKV's gains originate from the gate and compensation mechanisms, not from the shared-set selection policy required for merging on GQA models, and it justifies applying the same union rule to the eviction baselines reported in Table 1 and Table 3.

I Hyperparameter Sensitivity (Observation Window and Kernel)

Table 8 varies the observation window size W and smoothing kernel size K on Gemma-2 (6-dataset subset, 25% KV retention).

Table 8: Sensitivity to observation window W and kernel size K on Gemma-2-9B-IT (6 datasets, 25% KV retention). Default: W=32, K=5.

W	K	Qasp	SAMs	HpQA	LCC	TREC	PRet	**Avg**
16	5	40.16	43.04	**47.42**	72.29	67.00	33.00	50.49
32	**5**	40.01	42.89	46.67	72.24	67.00	**33.50**	50.39
64	5	39.71	**43.24**	**47.50**	72.07	66.50	33.00	50.34
32	3	**40.15**	42.74	46.90	71.98	**70.59**	**33.50**	**50.98**
32	7	39.83	42.73	46.50	**72.50**	67.50	33.00	50.34

All configurations fall within a 0.6-point range, indicating robustness to these hyperparameter choices.

J Category-Level Analysis

Table 9: Category-level averages on LongBench (Gemma-2-9B-IT, 25% KV retention). Best one-shot result per category in **bold**.

Category	**Full**	**SelKV(ours)**	**SnapKV**	**LOOK-M**	**PyramidKV**
Single-Doc QA	36.80	**36.59**	36.34	35.11	36.28
Multi-Doc QA	39.61	**38.80**	38.48	38.45	38.69
Summarization	23.97	**22.35**	22.07	21.96	21.84
Few-shot	67.66	**67.22**	67.01	63.31	66.93
Synthetic	20.25	**20.25**	20.25	20.25	20.25
Code	71.56	70.80	**71.19**	70.96	70.82

K Compression Ratio Sweep

Tables 10 and 11 present per-method scores at six compression ratios. On LLaMA-3.1, SnapKV leads at 10%; from 25% onward, SelKV matches or exceeds SnapKV; above 70%, all methods converge. On Gemma-2, SelKV leads at every budget from 25% onward.

Table 10: LongBench average (LLaMA-3.1-8B, 3.5k tokens).

Method	10%	25%	50%	70%	80%	90%
Full Cache	40.54					
SnapKV	38.06	39.79	40.29	40.41	40.49	40.46
LOOK-M	36.65	38.78	40.27	40.40	40.49	40.51
SelKV(ours)	37.56	39.81	40.34	40.38	40.42	40.46

Table 11: LongBench average (Gemma-2-9B-IT, 3.5k tokens).

Method	10%	25%	50%	70%	80%	90%
Full Cache	42.98					
SnapKV	**41.28**	42.16	42.69	42.87	42.92	**42.96**
LOOK-M	39.11	41.18	42.37	42.73	42.84	42.95
SelKV(ours)	41.05	**42.31**	**42.80**	**42.93**	**43.01**	**42.96**

We flag the small excesses over full cache at 80% retention on Gemma-2 (43.01 vs. 42.98) as consistent with, but not strong independent evidence for, the multi-document QA pattern in Section 4.2; individual deltas of this size are within the noise band established in Section 4.3.

L Long-Context Evaluation (31,500 Tokens)

Table 12: LongBench average (LLaMA-3.1-8B, 31,500 tokens, 25% KV retention).

Method	Avg (16 ds)	Δ
Full Cache	49.36	–
SnapKV	48.46	−0.90
PyramidKV	48.26	−1.10
LOOK-M	47.16	−2.20
SelKV (ours)	**48.46**	−0.90

M Implementation Details

Hardware. All experiments ran on NVIDIA H100 80GB GPUs, CUDA 12.1, PyTorch 2.4.

Statistical methodology. Paired bootstrap resampling was run with 10,000 iterations over per-example scores within each dataset, aggregated across datasets, with a fixed seed (42) for reproducibility. Because decoding is greedy and deterministic (temperature 0, no sampling), the only source of variance is the finite evaluation sample, which paired bootstrap over per-example scores directly captures. All configurations being compared in a given test share identical token selection, hyperparameters, and code path, varying only the component under test.

Evaluation. We follow the LongBench evaluation pipeline with greedy decoding. LongChat uses 31,500-token inputs; all other models use 3,500-token inputs. Code-completion tasks (LCC, RepoBench-P) use 500 samples; all other tasks use 200 samples.

Runtime. A full 16-dataset evaluation for one model takes approximately 1–2 hours on a single H100 GPU. The ratio-sweep ablation (6 ratios $\times$ 16 datasets $\times$ 3 methods per model) requires approximately 8 GPU-hours per model.

N Qualitative Generation Examples

The following examples compare generated text from LLaMA-3.1-8B-Instruct under Full Cache and SelKV (25% KV budget). All outputs use greedy decoding (temperature 0).

Example 1: Qasper (Single-Document QA). A scientific article about characterizing political fake news on Twitter (3,500 tokens) is provided as context. The question asks: *"How is the ground truth for fake news established?"*

Ground truth: Ground truth is not established in the paper.

Full Cache

[...] who found that fake news items are more likely to be shared than legitimate articles coming from trusted sources, and that they tend to exhibit a larger level of polarization. *Conclusion:* In this paper, we have presented a preliminary characterization of fake news in Twitter by looking into meta-data embedded in tweets. Our results show that there are specific pieces of meta-data about tweets that may allow the identification of fake news [...]

SelKV (25% KV budget)

[...] who found that fake news is more likely to be polarized than legit news. *Conclusion:* In this paper, we have characterized political fake news on Twitter by its meta-data. We found that tweets containing fake news are more likely to be created by unverified accounts, are shorter-lived, and are more likely to be polarized [...]

SelKV produces coherent, topically consistent summaries that preserve the key conclusions while differing only in phrasing.

Example 2: SAMSum (Dialogue Summarization). A dialogue dataset sample (3,500 tokens of few-shot examples) is provided. The target dialogue is between William and Emilia. **Ground truth:** Emilia is still angry.

Full Cache

Vicki will come to the place at 21:30 tomorrow night. They will be open until 22:00. [continues generating further dialogue-summary pairs]

SelKV (25% KV budget)

Vicki is asking about the store hours for tomorrow night. [continues generating further dialogue-summary pairs]

Both outputs demonstrate that the compressed cache retains sufficient context for in-context learning from few-shot examples, with SelKV producing slightly more concise summaries while preserving factual accuracy.