Behavior-Preserving KV Cache Compression

Doo Hwan Hwang[1], Junyoung Jang[1], Junho Na[1], Hosung Lim[2], Kee-Eung Kim[1]
[1]Kim Jaechul Graduate School of AI, KAIST, Daejeon, Republic of Korea
[2]KT Corporation, Seoul, Republic of Korea
Correspondence: dhhwang@ai.kaist.ac.kr

Abstract

KV caches are a major bottleneck in long-context inference and long-form generation with large language models. Existing training-free eviction policies largely rely on proxy importance signals, such as attention mass, to decide which past tokens to retain. We argue that cache compression should instead preserve the predictive behavior of the full-cache model, retaining entries whose removal would substantially change the model's output distribution. We propose Behavior-Preserving KV Cache Compression, a training-free framework that scores candidate evictions by estimating the compressed-cache logits induced by their removal and evaluating the resulting KL to the full-cache next-token distribution. Using pre-eviction forward statistics, the method avoids running separate masked forward passes for each candidate. Across diverse architectures and both prefill-time and generation-time compression, our method delivers substantial gains in downstream task quality over lightweight attention-based heuristics at matched retained-KV budgets, with the largest gains under aggressive compression. It achieves these gains with additional compression-time computation while retaining an end-to-end speedup over full-cache inference in our evaluated settings.

1 Introduction

Large language models (LLMs) rely on key-value (KV) caches during autoregressive decoding to avoid recomputing past activations (Zhang et al., 2023; Devoto et al., 2025). As sequence lengths grow, these caches scale linearly with the number of processed tokens and become a major bottleneck for memory usage and attention computation (Liu et al., 2023; Li et al., 2024; Zhang et al., 2023). KV cache compression reduces this cost by retaining only a subset of cached key-value entries. We focus on training-free cache management, where the eviction rule is computed from inference-time statistics rather than learned from task-specific traces or optimized through additional cache-policy training. This setting is attractive for deployment because it can be applied to off-the-shelf LLMs across different budgets, context lengths, and tasks without training an auxiliary eviction policy.

In this setting, the natural reference is the behavior of the same model before eviction. A compressed cache should therefore not merely satisfy a memory budget; it should preserve the predictive behavior of the pre-eviction computation. Since future decoding states are unavailable when compression decisions are made, we use the next-token distribution at available query positions as an observable reference behavior.

Existing training-free methods approximate this goal through indirect proxy signals. Attention-based methods retain entries that receive high attention in the current context (Liu et al., 2023; Li et al., 2024; Zhang et al., 2023) or are expected to receive high attention during future decoding (Devoto et al., 2025). Other methods favor entries that help reconstruct the original context from the compressed cache (Kim et al., 2025). These proxy objectives need not match the predictive effect of eviction: attention can overvalue redundant entries, while reconstruction utility can favor context recovery rather than preserving the next-token distribution. This mismatch motivates scoring cache entries by their estimated effect on model predictions.

We propose a **behavior-preserving framework** for KV cache compression. Our method scores candidate evictions by estimating their effect on the full-cache next-token distribution. It estimates the compressed-cache logits induced by eviction from pre-eviction forward statistics using a residual-path approximation, and plugs the estimated logits into a KL objective against the full-cache prediction. This avoids candidate-specific masked forward passes while ranking entries by their predicted behavioral effect.

We evaluate the method in two inference settings. In prefill-time compression, the prompt cache is compressed after long-context encoding. In generation-time compression, generated-token caches are compressed online during autoregressive decoding. Across models, datasets, and cache budgets, our method delivers substantially higher downstream task quality than lightweight attention-based eviction methods at matched retained-KV budgets, particularly under aggressive compression. Its behavior-preserving scoring incurs additional compression-time computation, while the resulting compressed inference remains faster than full-cache inference in our evaluated settings.

Our contributions are as follows:

- We recast training-free KV cache compression as behavior-preserving eviction, evaluating cached entries by their predicted effect on the full-cache next-token distribution rather than by proxy signals such as attention mass or reconstruction utility.
- We introduce a forward-only scoring method that estimates compressed-cache logits from pre-eviction forward statistics and evaluates their KL to the full-cache prediction, without candidate-specific masked forward passes.
- We provide a broad evaluation in both prefill-time and generation-time compression settings, showing that behavior-preserving scoring yields stronger compression–performance trade-offs, especially under aggressive cache budgets.

2 Related Work

KV cache compression reduces the memory and attention cost of long-context inference by reducing the stored KV states. Learned KV cache compression methods modify the model or train an auxiliary mechanism to improve the compression–quality trade-off. Dynamic Memory Compression learns head- and layer-specific compression ratios through continued pretraining (Nawrot et al., 2024), while Dynamic Memory Sparsification trains a sparse cache policy for inference-time scaling (Łańcucki et al., 2025). These approaches can be effective, but they require additional training or model retrofitting.

Training-free KV cache compression instead computes compression decisions from inference-time statistics. Early token-selection methods exploit attention persistence by retaining heavy hitters, recent tokens, or tokens that received high attention scores in previous decoding steps (Liu et al., 2023; Zhang et al., 2023). SnapKV extends this idea to prompt compression by selecting entries from attention patterns observed near the end of the prompt, while PyramidKV combines attention-based selection with layer-wise cache budgets (Li et al., 2024; Cai et al., 2025b). Orthogonal to token selection, KIVI compresses the KV cache by reducing numerical precision through tuning-free quantization (Liu et al., 2024).

Recent training-free KV cache compression methods use richer proxy objectives. Expected Attention estimates future attention from a closed-form model of future-query distributions (Devoto et al., 2025). KVzip scores cached entries by their contribution to reconstructing the original context, enabling query-agnostic compressed caches (Kim et al., 2025). CAOTE estimates the attention-output error caused by token removal (Goel et al., 2026), which captures an eviction-induced perturbation in attention-output space but not its resulting effect on the model's output distribution. R-KV combines token importance with redundancy for reasoning-model decoding (Cai et al., 2025a). These methods broaden KV cache compression beyond raw attention scores, but they still optimize proxy objectives such as future attention, reconstruction utility, attention-output error, or redundancy. In contrast, our method scores evictions by their estimated effect on the full-cache next-token distribution, aligning cache retention with predictive-behavior preservation rather than proxy importance.

3 Motivation and Analysis

3.1 Behavior Preservation at Available Query Positions

KV cache compression aims to reduce memory and attention cost while preserving the predictive behavior of the full-cache computation. Because a decoder-only language model defines a distribution over continuations through a product of next-token distributions, preserving behavior over a future continuation can be stated as preserving the autoregressive distribution induced by the compressed cache.

Let $y_{\leq t}$ denote the prefix observed up to position t. For a future continuation length τ, let P_t^τ and Q_t^τ denote the conditional distributions over $y_{t+1:t+\tau}$ induced by the full and compressed KV caches, respectively. A natural behavior-preserving

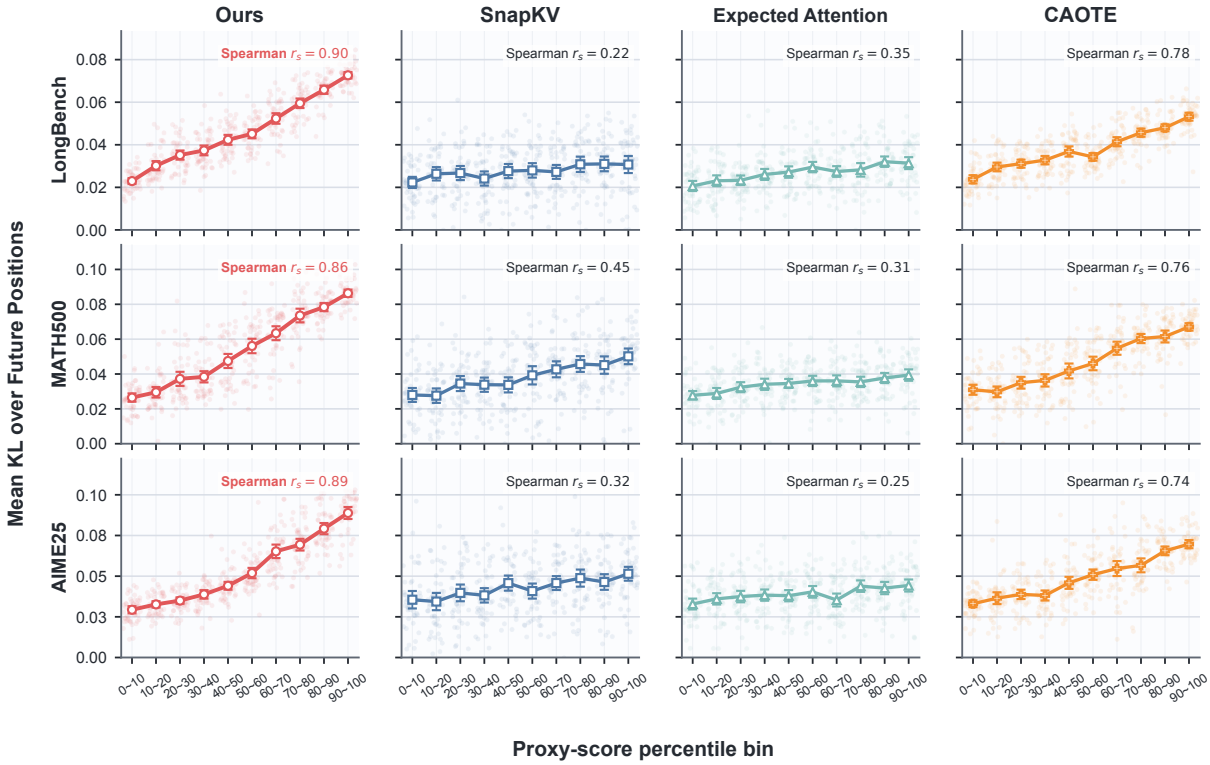

Figure 1: Relationship between method-specific proxy scores and KL over future positions on LongBench, MATH500, and AIME25 using LLaMA3.1-8B. For each scoring rule, we group candidate eviction masks by the percentile of the method's own score and sample perturbed KV caches from each score bin. We then measure the mean KL over future positions along the same reference continuation. The x-axis reports the score percentile. Higher percentiles should correspond to larger KL over future positions if a score captures the predictive effect of eviction. The behavior-preserving score exhibits the clearest association, while the association is less pronounced for proxy-based scores.

objective is

$$D_{\mathrm{KL}}(P_t^{\tau} \,\|\, Q_t^{\tau})\,.$$

Let p_t and q_t be the corresponding next-token distributions at the current prefix. By the chain rule for autoregressive KL,

$$D_{\mathrm{KL}}(P_t^{\tau} \,\|\, Q_t^{\tau}) = D_{\mathrm{KL}}(p_t \,\|\, q_t) + \mathbb{E}_{y_{t+1}\sim p_t}\left[D_{\mathrm{KL}}\left(P_{t+1}^{\tau-1} \,\middle\|\, Q_{t+1}^{\tau-1}\right)\right].$$

Thus, the next-token KL is the first term of the continuation-level behavioral discrepancy. If compression substantially changes this term, then it has already changed the model's predictive behavior at the current prefix, independently of how later states evolve.

The remaining terms depend on future tokens and future query states, which are not available when compression is applied. Estimating them would require rollouts, a learned model of future queries, or task-specific assumptions. We therefore use the observable component of the behavior-preserving objective. For the set of query positions $\mathcal{T}$ available at compression time, we define

$$\mathcal{J} = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} D_{\mathrm{KL}}(p_t \,\|\, q_t)\,.$$

This quantity measures compression error directly in output-distribution space and can be evaluated before future queries are realized.

Next-token KL does not fully characterize future behavior, since later queries may amplify or dampen the perturbation introduced by compression. Nevertheless, it is the observable component of the long-horizon behavioral divergence at compression time. Figure 1 supports using an output-distribution-based compression score: using LLaMA3.1-8B, we group sampled eviction masks by each method's score percentile and measure the mean KL over subsequent positions along the same reference continuation. Higher score percentiles should correspond to larger future-position KL if the score captures the predictive effect of eviction. The behavior-preserving score shows the clearest association, while the association is less

pronounced for proxy-based scores. We therefore use next-token KL as the compression-time behavioral target and score evictions by their estimated output-distribution perturbation.

4 Behavior-Preserving KV Cache Compression

4.1 Behavior-Preserving Objective

Let the pre-eviction KV cache contain S token positions across L Transformer layers. We use a cache-retention mask $d \in \{0,1\}^{L\times S}$, where $d_{\ell j} = 1$ retains token j's KV entry at layer ℓ, and $d_{\ell j} = 0$ evicts it. We also define the corresponding eviction indicator $\bar{d}_{\ell j} = 1 - d_{\ell j}$. The mask is shared across KV heads within each layer.

As discussed in Section 3.1, we use the next-token distribution at available query positions as the compression-time behavioral target. Let $\mathcal{T}$ denote these positions: prompt-end positions for prefill-time compression and recent generated positions for generation-time compression. For each $t \in \mathcal{T}$, let z_t be the full-cache vocabulary logits and $p_t = \mathrm{softmax}(z_t)$. If a retention mask d induces the exact logit perturbation $\Delta z_{t,d}$, the compressed-cache prediction is

$$q_t^{(d)} = \mathrm{softmax}(z_t + \Delta z_{t,d}).$$

The ideal mask minimizes the KL from the full-cache prediction to the compressed-cache prediction:

$$\mathcal{J}(d) = \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} D_{\mathrm{KL}}\left(p_t \,\middle\|\, q_t^{(d)}\right). \tag{1}$$

This objective makes the target explicit, but it is not directly usable for selection because obtaining $\Delta z_{t,d}$ for many candidate masks would require candidate-specific masked-cache forward passes. We instead estimate $\Delta z_{t,d}$ from full-cache forward statistics by first computing layerwise perturbations and then propagating them to the logits.

4.2 Computing Layer Perturbations from KV Eviction

We first estimate how a layer-level cache mask changes the attention output in the residual stream. This gives a layerwise residual perturbation $\Delta u_{\ell t}(d_\ell)$, which will later be propagated to the vocabulary logits. For layer ℓ, attention head r, and query position $t \in \mathcal{T}$, let $a_{\ell rtj}$ be the pre-eviction attention weight assigned to token j. Positions not visible under the causal mask have attention weight zero. Let $W^O_{\ell r}$ denote the block of the layer-ℓ attention output projection corresponding to head r. We write

$$\tilde{v}_{\ell rj} = W^O_{\ell r} v_{\ell rj}, \quad o_{\ell rt} = \sum_{i=1}^{S} a_{\ell rti}\tilde{v}_{\ell ri},$$

where $\tilde{v}_{\ell rj}$ is token j's output-projected value and $o_{\ell rt}$ is the full-cache projected head contribution.

For a layer mask d_ℓ, we use a deletion counterfactual that holds the pre-eviction query, keys, and values fixed, removes masked entries, and renormalizes attention over the retained entries. The resulting projected head-output perturbation is

$$\Delta o_{\ell rt}(d_\ell) = \frac{\sum_{i=1}^{S} d_{\ell i} a_{\ell rti}\tilde{v}_{\ell ri}}{\sum_{i=1}^{S} d_{\ell i} a_{\ell rti}} - o_{\ell rt}. \tag{2}$$

The denominator is the retained attention mass, so Eq. (2) is the difference between the renormalized retained-cache head contribution and the full-cache head contribution.

Summing over heads gives the residual-stream perturbation injected by the attention block at layer ℓ:

$$\Delta u_{\ell t}(d_\ell) = \sum_{r=1}^{H} \Delta o_{\ell rt}(d_\ell). \tag{3}$$

Equations (2) and (3) are computed from full-cache attention weights and cached values. They define the deletion-induced residual perturbations that are propagated to logits in the next subsection.

4.3 From Layer Perturbations to Logit Perturbations

The logits z_t in Section 4.1 are produced from the final residual state of the Transformer. Let x_t^L be the full-cache final pre-normalization residual state at position t. The corresponding logits are

$$z_t = W N_f(x_t^L) + b,$$

where N_f is the final normalization, W is the LM head, and b is the LM-head bias if present. If a mask d changes the final residual state by $\Delta x^L_{t,d}$, then a first-order expansion around the full-cache state gives

$$\Delta z_{t,d} \approx W J_t \Delta x^L_{t,d}, \quad J_t = \nabla N_f(x_t^L). \tag{4}$$

Computing $\Delta x^L_{t,d}$ exactly would require an additional forward computation under the masked cache. Let $\Delta u_{\ell t}(d_\ell)$ denote the residual-stream

perturbation injected by the attention block at layer ℓ under the layer mask d_ℓ. Since attention outputs are added to the residual stream, we approximate the final-state perturbation by the direct residual-path contribution:

$$\widehat{\Delta x}^{L}_{t,d} = \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell). \tag{5}$$

This approximation keeps the component computable from pre-eviction statistics and treats the responses of later attention, MLP, and normalization layers as approximation error. We validate this residual-path ranking signal in Appendix B.

Substituting Eq. (5) into Eq. (4) gives the estimated logit perturbation

$$\widehat{\Delta z}_{t,d} = W J_t \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell). \tag{6}$$

We then form the estimated compressed-cache logits

$$\hat{z}_{t,d} = z_t + \widehat{\Delta z}_{t,d}. \tag{7}$$

Plugging these logits into Eq. (1) gives the practical behavior-preserving score:

$$\begin{aligned} \widehat{\mathcal{J}}(d) &= \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} D_{\mathrm{KL}}\left(p_t \,\middle\|\, \hat{q}_t^{(d)}\right), \\ \hat{q}_t^{(d)} &= \mathrm{softmax}(\hat{z}_{t,d}). \end{aligned} \tag{8}$$

Small values of $\widehat{\mathcal{J}}(d)$ indicate masks whose estimated compressed-cache logits preserve the full-cache predictive distribution.

4.4 Budgeted Mask Selection

We select B layer-token entries to evict. Let $\mathcal{P}$ denote protected entries that must be retained, and let $\mathcal{E}$ denote the eligible entries. Using the eviction indicator $\bar{d}_{\ell j} = 1 - d_{\ell j}$, the feasible masks are

$$\mathcal{D}_B = \Big\{ d \in \{0,1\}^{L \times S} \,\Big|\, \bar{d}_{\ell j} = 0 \; \forall (\ell, j) \in \mathcal{P}, \sum_{(\ell,j) \in \mathcal{E}} \bar{d}_{\ell j} = B \Big\}.$$

The score in Eq. (8) evaluates the behavioral effect after applying a retention mask, but directly minimizing it over $\mathcal{D}_B$ would require searching over all subsets of B evicted entries. This is intractable because eviction costs are non-additive: removing multiple entries changes both the renormalized attention outputs and the resulting predictive distribution. We therefore use a two-stage greedy procedure.

First, we form a candidate pool using singleton scores. For an eligible entry (ℓ, j), we consider the singleton eviction indicated by $\bar{d}_{\ell j} = 1$, while all other eligible eviction indicators remain zero. The corresponding single-entry logit estimate is

$$\hat{z}_{t,(\ell,j)} = z_t + W J_t \Delta u_{\ell t}\big(\bar{d}_{\ell j}\big).$$

We score this singleton eviction by plugging the estimated logits into the behavior-preserving KL:

$$\frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} D_{\mathrm{KL}}\big(p_t \,\|\, \mathrm{softmax}(\hat{z}_{t,(\ell,j)})\big).$$

We construct $\mathcal{C}_{\mathrm{pool}}$ from the eligible entries with the lowest singleton-eviction scores.

Second, we refine the selected evictions using mask-level marginal costs. Starting from an empty selected set $\mathcal{M}$, we repeatedly add the candidate in $\mathcal{C}_{\mathrm{pool}}$ that causes the smallest increase in the estimated-logit KL score from Eq. (8), updating cached mask statistics after each step. We continue until B entries have been selected. This refinement accounts for interactions among evicted entries, including attention renormalization and changes in the estimated compressed-cache distribution. Implementation details, including top-K KL evaluation and refinement batching, are given in Appendix C.

5 Experiments

5.1 Experimental Setup

Models and datasets. We evaluate KV cache compression in two inference settings. In *prefill-time compression*, the prompt KV cache is compressed after long-context encoding and before answer generation. This setting tests whether compression preserves task-relevant information stored in the prompt cache. Since compression is applied after the full prompt forward pass, it reduces the cache used for subsequent generation but not the compute or transient memory required for initial prompt prefill. We evaluate Qwen3-8B (Yang et al., 2025), LLaMA3.1-8B (Grattafiori et al., 2024), and Gemma3-12B (Gemma Team, 2025) on the full LongBench benchmark (Bai et al., 2024). We also evaluate on RULER at 4K, 8K, and 16K context lengths to test controlled long-context retrieval and aggregation (Hsieh et al., 2024).

Model	Metric	Method	Full	$\rho=0.10$	$\rho=0.25$	$\rho=0.50$	$\rho=0.75$	$\rho=0.90$
Qwen3-8B	LongBench Avg.	SnapKV	48.9	47.5	47.0	45.1	40.7	32.2
		PyramidKV		47.4	46.8	44.7	40.0	33.0
		KVzip		47.7	47.2	45.9	42.7	36.1
		EA		48.0	47.6	47.2	45.8	38.3
		CAOTE		47.8	47.6	47.0	44.4	37.2
		Ours		**48.2**	**48.0**	**47.9**	**47.4**	**41.6**
	RULER 8K	SnapKV	86.1	85.4	84.5	78.8	62.7	47.9
		PyramidKV		85.6	84.3	80.6	64.8	49.3
		KVzip		85.7	85.0	81.8	66.2	52.4
		EA		85.5	85.2	80.7	69.4	54.0
		CAOTE		85.8	84.9	82.3	70.1	56.7
		Ours		**86.0**	**85.6**	**83.0**	**75.4**	**66.2**
	RULER 16K	SnapKV	79.5	78.7	77.4	70.8	54.9	37.1
		PyramidKV		79.0	77.7	72.5	56.4	39.6
		KVzip		78.9	77.8	73.4	60.4	45.2
		EA		79.1	77.6	72.6	59.4	44.2
		CAOTE		79.0	77.9	73.2	61.2	46.7
		Ours		**79.3**	**78.0**	**73.5**	**68.8**	**56.7**
LLaMA3.1-8B	LongBench Avg.	SnapKV	47.3	45.8	45.2	43.4	38.2	29.5
		PyramidKV		45.8	45.3	43.7	39.2	31.6
		KVzip		46.1	45.8	44.5	40.9	35.4
		EA		46.4	46.5	46.9	43.1	36.9
		CAOTE		46.3	46.2	46.4	43.4	37.8
		Ours		**46.6**	**47.0**	**47.0**	**44.8**	**41.8**
	RULER 8K	SnapKV	80.2	79.3	78.0	71.5	56.8	41.2
		PyramidKV		79.6	78.7	73.9	58.1	44.0
		KVzip		79.8	78.6	74.3	61.2	46.8
		EA		79.7	79.1	73.5	63.7	49.6
		CAOTE		79.9	78.9	75.2	63.1	51.0
		Ours		**80.0**	**79.5**	**76.0**	**73.8**	**61.2**
	RULER 16K	SnapKV	75.0	74.0	72.3	64.1	45.8	28.8
		PyramidKV		74.5	72.9	66.2	47.1	31.6
		KVzip		74.4	73.1	66.9	50.4	33.8
		EA		74.6	72.7	67.5	51.0	36.2
		CAOTE		74.3	73.6	68.0	56.1	40.6
		Ours		**74.8**	**73.8**	**70.5**	**66.3**	**52.5**
Gemma3-12B	LongBench Avg.	SnapKV	51.2	50.0	49.5	47.7	43.2	34.8
		PyramidKV		50.1	49.8	48.1	44.0	36.2
		KVzip		50.3	49.9	48.4	44.8	38.0
		EA		51.0	50.4	48.1	41.2	32.5
		CAOTE		50.5	50.0	49.0	46.3	40.2
		Ours		**51.2**	**50.8**	**50.0**	**48.4**	**43.5**
	RULER 8K	SnapKV	88.0	87.3	86.0	79.2	63.8	45.8
		PyramidKV		87.5	86.6	81.3	65.4	47.9
		KVzip		87.4	86.8	81.0	68.2	49.7
		EA		87.7	86.5	82.2	69.6	52.4
		CAOTE		87.6	87.0	82.9	71.0	55.8
		Ours		**87.8**	**87.2**	**83.8**	**76.0**	**68.0**
	RULER 16K	SnapKV	80.0	79.2	77.8	71.4	55.6	38.7
		PyramidKV		79.5	78.4	72.9	56.5	42.6
		KVzip		79.4	78.6	74.2	63.1	49.2
		EA		79.6	78.2	73.2	62.6	47.2
		CAOTE		79.3	78.8	74.6	62.8	49.6
		Ours		**79.8**	**79.0**	**75.5**	**68.0**	**60.0**

Table 1: Aggregate prefill-time compression results across three model families. We report LongBench average and RULER results at 8K and 16K context lengths as a function of the compression ratio ρ, where ρ denotes the fraction of evicted KV entries. Detailed task-level results for LLaMA3.1-8B and RULER 4K results are provided in Appendix D. **Bold** denotes the best score within each model–metric block, excluding the full-cache reference. EA denotes Expected Attention.

Model	Data	Method	Maximum generation length											
			2048				4096				8192			
			No comp.	2×	4×	12×	No comp.	2×	4×	12×	No comp.	2×	4×	12×
DeepSeek-R1 Qwen-7B	MATH500	SnapKV	81.6	80.6	77.2	64.8	84.0	83.1	78.4	60.2	84.2	83.3	78.0	58.8
		PyramidKV		80.4	76.8	63.9		82.9	78.0	59.6		83.0	77.5	57.9
		EA		80.9	78.7	68.0		83.4	80.0	65.0		83.2	80.2	63.4
		CAOTE		81.0	79.0	67.6		83.6	80.4	66.8		83.7	80.6	65.2
		R-KV		**81.2**	80.1	**76.6**		**83.8**	82.2	78.6		**84.0**	82.8	79.2
		Ours		81.1	**80.5**	**76.6**		83.7	**82.8**	**79.8**		**84.0**	**83.0**	**80.4**
	AIME25	SnapKV	17.8	15.6	12.2	7.8	23.3	20.0	16.7	8.9	27.8	23.3	17.8	10.0
		PyramidKV		15.6	12.2	7.8		20.0	15.6	8.9		23.3	17.8	10.0
		EA		16.7	13.3	8.9		21.1	17.8	11.1		24.4	20.0	12.2
		CAOTE		15.6	14.4	10.0		22.2	18.9	13.3		25.6	21.1	14.4
		R-KV		**17.8**	16.7	**14.4**		**23.3**	21.1	**18.9**		**27.8**	25.6	**21.1**
		Ours		**17.8**	**17.8**	13.3		22.2	**22.2**	17.8		26.7	**26.7**	20.0
Qwen2.5 Math-7B	MATH500	SnapKV	79.8	79.0	75.9	60.8	82.5	81.3	76.8	56.8	82.9	81.4	76.3	55.2
		PyramidKV		78.8	75.6	59.9		81.1	76.4	56.0		81.2	75.8	54.6
		EA		79.1	76.8	63.8		81.7	78.2	61.8		81.8	78.1	60.3
		CAOTE		79.3	77.6	65.4		82.0	79.0	63.0		82.0	78.8	62.1
		R-KV		**79.6**	78.9	72.8		82.2	80.6	76.5		**82.4**	81.0	77.0
		Ours		79.5	**79.3**	**74.0**		**82.3**	**81.2**	**78.0**		82.3	**81.4**	**79.0**
	AIME25	SnapKV	11.1	8.9	6.7	3.3	15.6	12.2	8.9	4.4	18.9	13.3	10.0	5.6
		PyramidKV		8.9	6.7	3.3		12.2	8.9	4.4		14.4	10.0	5.6
		EA		10.0	7.8	4.4		13.3	10.0	6.7		15.6	12.2	7.8
		CAOTE		10.0	8.9	5.6		14.4	11.1	7.8		16.7	13.3	8.9
		R-KV		**11.1**	10.0	**7.8**		**15.6**	13.3	**11.1**		**18.9**	16.7	**13.3**
		Ours		10.0	**11.1**	6.7		14.4	**14.4**	10.0		17.8	**17.8**	**13.3**

Table 2: Generation-time compression results under different maximum generation lengths. Scores are mean pass@1 over three independent stochastic runs. For $n \in \{2, 4, 12\}$, $n\times$ denotes that the eligible generated-token KV cache is kept approximately n times smaller. **Bold** denotes the best score within each setting, excluding the no-compression reference.

In *generation-time compression*, the prompt cache is kept intact and compression is applied online to generated-token KV entries during decoding. This setting tests whether cache growth during long reasoning can be controlled without disrupting generation. We evaluate DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025) and Qwen2.5-Math-7B (Yang et al., 2024) on MATH500 (Lightman et al., 2024) and AIME25 (Mathematical Association of America, 2024–2025), which require multi-step mathematical reasoning.

Baselines. We compare with representative training-free KV cache compression methods: SnapKV and PyramidKV for attention-based token selection (Li et al., 2024; Cai et al., 2025b), KVzip for query-agnostic context reconstruction (Kim et al., 2025), Expected Attention for future-attention estimation (Devoto et al., 2025), and CAOTE for attention-output error scoring (Goel et al., 2026). We also include the no-compression full-cache model as the reference. For generation-time compression, we additionally compare with R-KV, a reasoning-specific KV cache compression baseline (Cai et al., 2025a). KVzip is evaluated only in prefill-time compression, as it targets query-agnostic prompt-cache reconstruction rather than online pruning.

Implementation and budget matching. All methods are evaluated with the same models, datasets, decoding settings, compression timing, and matched final retained layer–KV-head–token budgets. We keep each baseline's native pruning granularity, protection rule, and layer-budget allocation, but protected entries always count toward the retained budget. Thus, no method receives free protected cache, and ρ denotes the evicted fraction under this accounting. For our method, each layer-token decision is shared across KV heads and is counted after expansion to the corresponding KV-head entries. Appendix C gives further details.

5.2 Prefill-time Compression

Table 1 summarizes prefill-time compression results across models, benchmarks, and context

lengths. The table reports LongBench average performance together with RULER results at 8K and 16K context lengths under matched cache budgets. As the compression ratio ρ increases, proxy-based baselines degrade more rapidly, whereas our method preserves performance more effectively. This pattern is largely consistent across the three model families and becomes especially clear under aggressive compression and longer-context RULER settings.

The results also show the benefit of evaluating evictions in predictive space rather than relying only on intermediate proxy signals. CAOTE measures eviction-induced error in attention-output space. In contrast, our method propagates the estimated residual-stream perturbation through the final normalization and LM head, forms estimated compressed-cache logits, and evaluates their KL to the full-cache next-token distribution. This predictive-space scoring gives modest gains at mild compression, but becomes more beneficial when the retained cache budget is tight, especially on RULER 8K and 16K.

5.3 Generation-time Compression

Unlike prefill-time compression, generated-token pruning is applied repeatedly as decoding proceeds. Longer generation horizons therefore create larger caches and more pruning events, making errors from poor eviction decisions more likely to accumulate over the reasoning process. Table 2 reports pass@1 at different maximum generation lengths under full-cache decoding and $2\times$, $4\times$, and $12\times$ generated-token cache compression.

Mild compression causes only small performance changes, but tighter budgets and longer generation horizons make the choice of eviction rule more consequential. Attention-based methods and CAOTE degrade more rapidly under these settings, while R-KV and our method remain more stable. R-KV is a strong reasoning-specific decoding-time baseline and is best in some settings. It uses an explicit importance–redundancy trade-off for reasoning traces, whereas our method uses the same estimated-logit KL score as in prefill-time compression without reasoning-specific redundancy heuristics. Its competitive generation-time performance suggests that the proposed behavior-preserving score transfers beyond one-shot prompt-cache compression to repeated online pruning of generated-token caches.

At $12\times$ compression, increasing the maximum generation length from 2048 to 8192 increases SnapKV's MATH500 degradation relative to full-cache decoding from 16.8 to 25.4 points on DeepSeek-R1-Distill-Qwen-7B and from 19.0 to 27.7 points on Qwen2.5-Math-7B, whereas our degradation remains within 3.8–5.8 points across these settings.

Appendix E quantifies the quality–efficiency trade-off. At matched retained-KV budgets, our method improves task quality at a higher compression-time cost than lightweight heuristics. In cross-budget comparisons, it retains 2.5–$3\times$ fewer KV entries than SnapKV while achieving higher task scores and remaining faster than full-cache inference.

6 Ablation Study

We ablate the main scoring and selection components in Appendix F.

Final-normalization propagation. The ablated variant treats the final normalization as identity, using $W \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell)$ instead of $W J_t \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell)$ to estimate the logit perturbation. The drop across metrics confirms that final normalization changes how residual-stream perturbations translate into compressed-cache logits and, consequently, into output-distribution changes.

Scoring window size. Using a single query position gives a noisier estimate of the behavioral effect of eviction, whereas eight positions provide little improvement over four. We therefore use $m_{\text{pre}} = m_{\text{gen}} = 4$ by default.

Top-K KL evaluation. We evaluate the estimated-logit KL score on the renormalized top-K support of the full-cache distribution. The small gains from increasing K suggest that a compact output support captures most of the predictive signal while keeping scoring efficient.

Selection refinement. Pool-refine uses singleton scores only to form a candidate pool, and selects final evictions by reevaluating the marginal KL cost under the current multi-entry mask. Unlike fixed additive scores, this refinement accounts for mask-level interactions caused by attention renormalization and changes in the estimated compressed-cache distribution. Its gains over singleton selection suggest that these interactions matter under tight cache budgets.

7 Conclusion

We presented Behavior-Preserving KV Cache Compression, a training-free framework that recasts KV eviction as preserving the predictive behavior of the full-cache model rather than optimizing proxy importance signals. Our method scores candidate evictions by estimating the compressed-cache logits induced by their removal and evaluating their KL to the full-cache next-token distribution, using only pre-eviction forward statistics and no candidate-specific masked forward passes. Across both prefill-time and generation-time compression, behavior-preserving scoring maintains task performance under tight cache budgets, supporting output-distribution preservation as an effective principle for training-free KV cache compression.

Limitations

A common challenge in training-free KV cache compression is that eviction decisions must be made before the future decoding queries that will read the cache are known. Our method follows this setting and uses the next-token distributions at available query positions as the observable reference behavior. This design preserves behavior only with respect to the available compression-time queries, and does not provide a guarantee for arbitrary unseen future continuations, where later queries may amplify or dampen the effect of eviction. Developing compression-time signals that better anticipate unseen future queries remains an important direction for KV cache compression.

Acknowledgments

This work was the result of project supported by KT(Korea Telecom). This work was also supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT): No. RS-2024-00457882 (AI Research Hub Project), RS-2019-II190075 (Artificial Intelligence Graduate School Program (KAIST)), No. RS-2024-00397310 (Development of an AI Simulator for Creating Transparent Compounds that Can Be Altered for Tactile Sensation), and No. RS-2024-00343989 (Enhancing the Ethics of Data Characteristics and Generation AI Models for Social and Ethical Learning). This work was supported by the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT) (IITP-2026-RS-2024-00436857). This research was supported by the "Advanced GPU Utilization Support Program" funded by the Government of the Republic of Korea (Ministry of Science and ICT).

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. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. 2025a. R-KV: Redundancy-aware KV cache compression for reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025b. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *Preprint*, arXiv:2406.02069.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. 2025. Expected attention: KV cache compression by estimating attention from future queries distribution. *CoRR*, abs/2510.00636.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, and 6 others. 2021. A mathematical framework for transformer circuits. `https://transformer-circuits.pub/2021/framework/index.html`.

Gemma Team. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew J. Morse, Matthew Harper Langston, Christopher Lott, and Mingu Lee. 2026. CAOTE: Optimizing KV cache memory through attention output error-based token eviction. In *Workshop on Memory for LLM-Based Agentic Systems at ICLR 2026*.

Aaron Grattafiori and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models?

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. 2025. KVzip: Query-agnostic KV cache compression with context reconstruction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Ponti. 2025. Inference-time hyper-scaling with KV cache compression. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning*.

Mathematical Association of America. 2024–2025. American Invitational Mathematics Examination (AIME).

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. 2024. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *ArXiv*.

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. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

A Derivation of the Layer Perturbations from KV Eviction

Fix a layer ℓ, head $r \in \{1, \ldots, H\}$, and scoring query position $t \in \mathcal{T}$. Let $\mathcal{I}_t \subseteq \{1, \ldots, S\}$ denote the cache positions visible to query t under the causal mask. Equivalently, one may sum over all positions by setting $a_{\ell rti} = 0$ for positions not visible to t.

Recall that

$$\tilde{v}_{\ell rj} = W^O_{\ell r} v_{\ell rj}$$

is the output-projected value vector for token j at layer ℓ and head r. The full-cache projected head output is

$$o_{\ell rt} = \sum_{i \in \mathcal{I}_t} a_{\ell rti} \tilde{v}_{\ell ri}.$$

For a layer retention mask d_ℓ, let $\bar{d}_{\ell j} = 1 - d_{\ell j}$. The removed attention mass and removed projected contribution are

$$A_{\ell rt}(d_\ell) = \sum_{j \in \mathcal{I}_t} \bar{d}_{\ell j} a_{\ell rtj}, \qquad R_{\ell rt}(d_\ell) = \sum_{j \in \mathcal{I}_t} \bar{d}_{\ell j} a_{\ell rtj} \tilde{v}_{\ell rj}.$$

In the deletion counterfactual, the pre-eviction query, keys, and values at layer ℓ are held fixed. Removing masked entries deletes their attention terms and renormalizes the retained attention weights by the retained mass $1 - A_{\ell rt}(d_\ell)$. Assuming $1 - A_{\ell rt}(d_\ell) > 0$, the masked projected head output is

$$o^{(d_\ell)}_{\ell rt} = \sum_{i \in \mathcal{I}_t} \frac{d_{\ell i} a_{\ell rti}}{1 - A_{\ell rt}(d_\ell)} \tilde{v}_{\ell ri} = \frac{o_{\ell rt} - R_{\ell rt}(d_\ell)}{1 - A_{\ell rt}(d_\ell)}.$$

Therefore the head-level perturbation is

$$\Delta o_{\ell rt}(d_\ell) = o^{(d_\ell)}_{\ell rt} - o_{\ell rt} = \frac{A_{\ell rt}(d_\ell) o_{\ell rt} - R_{\ell rt}(d_\ell)}{1 - A_{\ell rt}(d_\ell)}.$$

Substituting the definitions of $A_{\ell rt}(d_\ell)$ and $R_{\ell rt}(d_\ell)$ gives the equivalent entrywise form

$$\Delta o_{\ell rt}(d_\ell) = \frac{\sum_{j \in \mathcal{I}_t} \bar{d}_{\ell j} a_{\ell rtj} \left(o_{\ell rt} - \tilde{v}_{\ell rj}\right)}{1 - A_{\ell rt}(d_\ell)}.$$

Summing over attention heads gives the residual-stream perturbation injected by the attention block at layer ℓ:

$$\Delta u_{\ell t}(d_\ell) = \sum_{r=1}^{H} \Delta o_{\ell rt}(d_\ell).$$

This expression is exact for the deletion counterfactual in which the pre-eviction query, keys, and values at layer ℓ are held fixed. Responses of later attention, MLP, and normalization layers are treated as approximation error in the main residual-path surrogate.

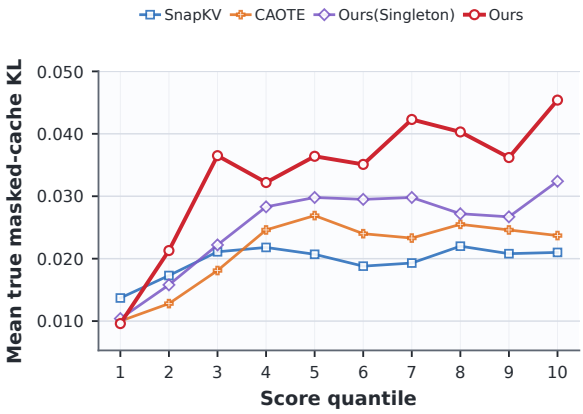

Figure 2: Validation of the residual-path approximation under multi-entry KV eviction on LLaMA3.1-8B using LongBench. We rank the same eviction masks by evicted SnapKV attention score, CAOTE attention-output error, our additive singleton KL score, and our mask-level estimated-logit KL score. For each score quantile, we report the mean true masked-cache KL $\mathcal{J}_{\text{true}}(d)$ from a masked-cache forward pass. The mask-level estimated-logit KL score gives the strongest ranking signal, with Spearman correlations of 0.45, 0.53, 0.67, and 0.77 for SnapKV, CAOTE, singleton, and mask-level estimated-logit KL, respectively.

B Validation of the Residual-Path Approximation

We validate whether the residual-path approximation provides a useful ranking signal under multi-entry KV eviction. This setting matches the actual compression procedure, where a mask removes many KV entries simultaneously. We use LLaMA3.1-8B on LongBench examples.

The residual-path approximation is motivated by the additive structure of Transformer residual connections, where attention blocks write updates into a shared residual stream (Vaswani et al., 2017; Elhage et al., 2021). We keep this directly written component as a tractable proxy for the candidate-induced final-state perturbation. This does not assume zero response from subsequent layers; those responses are treated as approximation error and evaluated empirically below.

For an eviction mask $d \in \mathcal{D}_B$, we compute the exact masked-cache behavioral change by running a masked-cache forward pass and measuring

$$\mathcal{J}_{\text{true}}(d) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} D_{\text{KL}}\left(p_t \,\middle\|\, q_t^{(d)}\right), \tag{9}$$

where $q_t^{(d)}$ is the predictive distribution after applying the mask. This quantity includes simultaneous KV removals, attention renormalization, and propagation through later layers, and serves as the masked-cache reference.

Our residual-path approximation estimates the logit perturbation as

$$\widehat{\Delta z}_{t,d} = W J_t \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell), \tag{10}$$

where $J_t = \nabla N_f(x_t^L)$ is the final-normalization Jacobian at the full-cache residual state, and $\Delta u_{\ell t}(d_\ell)$ is the residual-stream perturbation injected by layer ℓ. We then form the estimated compressed-cache logits

$$\hat{z}_{t,d} = z_t + \widehat{\Delta z}_{t,d}$$

and evaluate the same behavior-preserving KL objective on these estimated logits:

$$\begin{aligned} \widehat{\mathcal{J}}(d) &= \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} D_{\text{KL}}\left(p_t \,\middle\|\, \hat{q}_t^{(d)}\right), \\ \hat{q}_t^{(d)} &= \text{softmax}(\hat{z}_{t,d}). \end{aligned} \tag{11}$$

Unlike $\mathcal{J}_{\text{true}}(d)$, this score uses only full-cache statistics and requires no masked-cache forward pass. It differs from the true masked-cache KL only in replacing the exact compressed-cache logits with residual-path estimated logits.

To contextualize this approximation, we compare with three simpler signals. All validation scores are evaluated on the same layer-token eviction masks. When a baseline provides head-specific scores, we aggregate those scores over heads under the shared layer-token mask. Recall that $\bar{d}_{\ell j} = 1 - d_{\ell j}$ indicates whether entry (ℓ, j) is evicted.

The first signal is the evicted SnapKV attention score,

$$E_{\text{SnapKV}}(d) = \frac{1}{LH} \sum_{\ell=1}^{L} \sum_{r=1}^{H} \sum_{j=1}^{S} \bar{d}_{\ell j} s_{\ell r j}^{\text{SnapKV}}. \tag{12}$$

The second is the CAOTE attention-output error induced by the same mask,

$$E_{\text{CAOTE}}(d) = \frac{1}{|\mathcal{T}|LH} \sum_{t \in \mathcal{T}} \sum_{\ell=1}^{L} \sum_{r=1}^{H} \|\Delta C_{\ell r t}(d_\ell)\|_2 . \tag{13}$$

The third is an additive singleton variant of our estimated-logit KL score:

$$\begin{aligned} E_{\text{single}}(d) = \frac{1}{B|\mathcal{T}|} \sum_{\ell=1}^{L} \sum_{j=1}^{S} \sum_{t \in \mathcal{T}} \bar{d}_{\ell j} \\ \times D_{\text{KL}}\left(p_t \,\middle\|\, \text{softmax}(\hat{z}_{t,(\ell,j)})\right). \end{aligned} \tag{14}$$

where $\hat{z}_{t,(\ell,j)}$ denotes the estimated logits under the singleton eviction of entry (ℓ, j). These comparisons isolate the effect of moving from attention scores to attention-output errors, from local errors to predictive-distribution scoring, and from additive singleton evaluation to mask-level evaluation.

For each compression ratio ρ, we evaluate all scores on the same multi-entry masks. We rank masks by each score, divide them into quantiles, and report the mean exact masked-cache KL $\mathcal{J}_{\mathrm{true}}(d)$ in each quantile, together with Spearman rank correlation. Figure 2 shows that the mask-level estimated-logit KL score has the strongest monotonic relationship with exact masked-cache KL. This supports the residual-path approximation as a proxy for ranking multi-entry cache-retention masks.

C Implementation Details

Statistics collected from the full-cache pass. Our method uses statistics derived from the unmasked pre-eviction computation and does not run candidate-specific masked forward passes. For each scoring position $t \in \mathcal{T}$, we obtain the full-cache logits z_t, keep the top-K support $\mathcal{K}_t$ and the renormalized top-K probabilities used by the KL evaluation, and capture the final pre-normalization residual state x_t^L. For each layer, we recompute only the attention rows for $\mathcal{T}$ from cached keys and query states, storing $H \times |\mathcal{T}| \times S$ attention rows per layer rather than full $H \times S \times S$ attention maps. We also store the projected full-cache head outputs $o_{\ell r t}$.

The projected value

$$\tilde{v}_{\ell r j} = W_{\ell r}^{O} v_{\ell r j}$$

denotes the output-projected value contribution used in the derivation. We do not precompute or store $\tilde{v}_{\ell r j}$ for all layers, heads, and tokens. In implementation, these products are computed for candidate token chunks during scoring and refinement, and the block-level projected-value buffers are discarded after use.

Final-normalization Jacobian-vector product. Since the LM head is applied after final normalization, the residual-stream perturbation must be propagated through the final normalization. For RMSNorm-based models, this Jacobian-vector product can be computed in closed form. Let

$$N_f(x) = \gamma \odot \frac{x}{s(x)}, \qquad s(x) = \sqrt{\frac{\|x\|_2^2}{D} + \epsilon_{\mathrm{rms}}},$$

where D is the hidden dimension. For a perturbation $v \in \mathbb{R}^D$ at scoring position t, with $s_t = s(x_t^L)$, we compute

$$J_t v = \gamma \odot \left(\frac{v}{s_t} - \frac{x_t^L((x_t^L)^\top v)}{D s_t^3} \right).$$

For models without final normalization, we set $N_f = I$ and $J_t = I$. The LM-head bias, if present, does not appear in the perturbation because it cancels when taking logit differences.

Top-K KL evaluation. The practical score in Eq. (8) evaluates the KL between the full-cache prediction and the estimated compressed-cache prediction. Full-vocabulary evaluation is expensive, so we evaluate this score on the top-K support of the full-cache distribution. Let $\mathcal{K}_t$ be the top-K tokens under $p_t = \mathrm{softmax}(z_t)$, and define

$$\widetilde{p}_{t,i} = \frac{p_{t,i}}{\sum_{k \in \mathcal{K}_t} p_{t,k}}, \qquad i \in \mathcal{K}_t.$$

For a mask d, let

$$v_{t,d} = \sum_{\ell=1}^{L} \Delta u_{\ell t}(d_\ell)$$

be the residual-path perturbation before final normalization. Its induced top-K logit shift is

$$\delta_{t,i}(d) = W_i J_t v_{t,d}, \qquad i \in \mathcal{K}_t,$$

where W_i is the i-th row of the LM head. Since the estimated compressed-cache logits are $\hat{z}_{t,d} = z_t + \widehat{\Delta z}_{t,d}$, the estimated compressed-cache distribution restricted to $\mathcal{K}_t$ can be written as

$$\widetilde{q}_{t,i}^{(d)} = \frac{\widetilde{p}_{t,i} \exp(\delta_{t,i}(d))}{\sum_{k \in \mathcal{K}_t} \widetilde{p}_{t,k} \exp(\delta_{t,k}(d))}, \qquad i \in \mathcal{K}_t.$$

The implemented top-K behavior-preserving score is

$$\widehat{\mathcal{J}}_K(d) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} D_{\mathrm{KL}}\left(\widetilde{p}_t \,\middle\|\, \widetilde{q}_t^{(d)} \right). \tag{15}$$

Equivalently, this KL can be evaluated without explicitly forming $\widetilde{q}_t^{(d)}$:

$$D_{\mathrm{KL}}\left(\widetilde{p}_t \,\middle\|\, \widetilde{q}_t^{(d)} \right) = \log \sum_{i \in \mathcal{K}_t} \widetilde{p}_{t,i} \exp(\delta_{t,i}(d)) - \sum_{i \in \mathcal{K}_t} \widetilde{p}_{t,i} \delta_{t,i}(d).$$

We evaluate this expression using a numerically stable log-sum-exp. When $\mathcal{K}_t$ is the full vocabulary, Eq. (15) recovers the estimated-logit KL in Eq. (8).

Setting	m	K	α	b_{ref}	ϵ	Extra
Prefill-time	$m_{\text{pre}} = 4$	$K_{\text{pre}} = 128$	$\alpha_{\text{pre}} = 2$	$b_{\text{pre}} = 512$	10^{-4}	–
Generation-time	$m_{\text{gen}} = 4$	$K_{\text{gen}} = 256$	$\alpha_{\text{gen}} = 2$	$b_{\text{gen}} = 256$	10^{-4}	$I_{\text{comp}} = 512,\ w_{\text{recent}} = 4$

Table 3: Default implementation hyperparameters for prefill-time and generation-time compression. K is the top-K support size used for KL evaluation. Values are fixed within each regime and are not tuned per benchmark.

Efficient evaluation during refinement. During the refinement stage, we evaluate $\widehat{\mathcal{J}}_K(d_{\mathcal{M}})$ and $\widehat{\mathcal{J}}_K(d_{\mathcal{M}\cup\{a\}})$ using the full-cache attention weights and cached values, rather than running masked forward passes. Here, $d_{\mathcal{M}}$ denotes the mask that evicts the entries in the current selected set $\mathcal{M}$. For a tentative candidate $a = (\ell, j)$, only the layer-ℓ removed-mass and removed-contribution statistics change; the statistics for all other layers are reused. We then compute

$$\text{cost}(a; \mathcal{M}) = \widehat{\mathcal{J}}_K(d_{\mathcal{M}\cup\{a\}}) - \widehat{\mathcal{J}}_K(d_{\mathcal{M}}).$$

After selecting candidates in a refinement round, we update the current retention mask and its cached layerwise statistics.

Pool-and-refine mask construction. We form $\mathcal{C}_{\text{pool}}$ from the $\lceil \alpha B \rceil$ eligible candidates with the lowest singleton-eviction scores. Starting from $\mathcal{M} = \emptyset$, each refinement round evaluates $\text{cost}(a; \mathcal{M})$ for the remaining candidates in $\mathcal{C}_{\text{pool}}$, adds the lowest-cost $\min(b_{\text{ref}}, B - |\mathcal{M}|)$ candidates, and updates the cached mask statistics. The process stops when $|\mathcal{M}| = B$. The cases $b_{\text{ref}} = 1$ and $b_{\text{ref}} \geq B$ recover fully greedy refinement and one-shot selection, respectively.

Numerical stability. During scoring and refinement, we reject any candidate whose addition would make the retained attention mass too small for any scoring position, layer, or head. Let

$$A_{\ell rt}(d_\ell) = \sum_{i=1}^{S} (1 - d_{\ell i}) a_{\ell rti}$$

be the removed attention mass. Since the pre-eviction attention weights sum to one over visible positions, the retained attention mass is $1 - A_{\ell rt}(d_\ell)$. We reject a candidate if

$$1 - A_{\ell rt}(d_\ell) < \epsilon.$$

This avoids unstable attention renormalization in Eq. (2).

Grouped-query attention. For grouped-query or multi-query attention, $v_{\ell rj}$ denotes the value vector from the KV head associated with query head r. We compute Eq. (3) at the query-head level using the corresponding shared KV head, while the cache-retention mask remains shared across heads within each layer.

Default hyperparameters. We use the same behavior-preserving score in both compression regimes, but use separate implementation hyperparameters because the eligible candidates and compression schedules differ. In prefill-time compression, compression is applied once after prompt prefill, $\mathcal{T}$ contains the last m_{pre} prompt positions, and the budget is assigned over eligible prompt layer-token entries. In generation-time compression, prompt entries are always protected, compression is triggered every I_{comp} generated tokens, and only generated-token entries outside the recent protected window w_{recent} are eligible; $\mathcal{T}$ contains the most recent m_{gen} generated positions at each compression point.

Table 3 lists the top-K size used for KL evaluation, pool factor α, refinement batch size b_{ref}, and stability threshold ϵ. For $n\times$ generation-time compression, the eligible generated-token cache is kept approximately n times smaller than the uncompressed eligible cache. All hyperparameters are fixed within each regime and are not tuned per benchmark.

Baseline settings and budget matching. Unless otherwise stated, baseline hyperparameters follow the settings recommended in the original papers and their corresponding KVPress implementations.[1] We evaluate each method with its native scoring rule, pruning granularity, and protection rule, and match methods by the final retained layer–KV-head–token budget.

For a cache of length S, we compute the prefill-time compression ratio by counting retained layer–

[1] https://github.com/NVIDIA/kvpress

KV-head–token entries:

$$\rho = 1 - \frac{\sum_{\ell=1}^{L} \sum_{h=1}^{H_{\mathrm{kv}}} |\mathcal{S}_{\ell,h}^{\mathrm{keep}}|}{L H_{\mathrm{kv}} S}.$$

Here, $\mathcal{S}_{\ell,h}^{\mathrm{keep}}$ denotes the retained token positions at layer ℓ and KV head h. For head-wise methods, this set may differ across KV heads. For our method, the layer-token mask is shared across KV heads within each layer, so a retained or evicted token position applies to all corresponding KV-head entries in that layer. Protected entries are counted as retained entries and therefore consume cache budget; no method receives uncounted protected cache.

For our method, we use $n_{\mathrm{sink}} = 4$ in all prefill-time experiments. SnapKV and PyramidKV use their observation windows for attention-based scoring and retain the observation-window entries according to their KVPress implementations.

Method	Native granularity	Native protected set	Layer budget
Ours	Layer-token	Sink	Global
SnapKV	KV-head-token	Observation window	Uniform
PyramidKV	KV-head-token	Observation window	Pyramidal
Expected Attn.	KV-head-token	Sink	Uniform
KVzip	KV-head-token	Sink	Global
CAOTE	KV-head-token	Sink	Global

Table 4: Implementation details for prefill-time compression. Baselines are evaluated with their original pruning granularity, protection rule, and layer-budget allocation. Protected entries are included in the matched final retained layer–KV-head–token budget for all methods, so no method receives uncounted protected cache.

D Experiment Results

Task-level prefill-time compression results

Task	Method	Full	$\rho=0.10$	$\rho=0.25$	$\rho=0.50$	$\rho=0.75$	$\rho=0.90$
Single-document QA							
NarrativeQA	SnapKV	28.8	27.2	27.3	25.5	22.9	17.5
	PyramidKV		27.3	27.4	25.9	23.4	19.3
	KVzip		27.7	27.5	26.6	24.8	21.0
	EA		27.7	**28.3**	**28.4**	26.2	22.6
	CAOTE		27.7	27.6	28.1	26.9	23.0
	Ours		**28.2**	28.1	28.2	**27.4**	**25.4**
Qasper	SnapKV	36.0	34.6	34.3	33.3	30.1	24.7
	PyramidKV		34.4	34.6	33.3	30.3	26.8
	KVzip		35.0	34.9	34.3	31.6	28.4
	EA		**35.6**	35.0	**36.0**	33.2	29.6
	CAOTE		35.1	35.4	35.7	33.8	30.3
	Ours		35.4	**35.6**	35.8	**34.2**	**32.8**
MultiFieldQA-en	SnapKV	55.0	53.6	53.3	51.8	48.8	43.7
	PyramidKV		53.6	53.1	52.2	49.4	45.7
	KVzip		54.2	53.5	52.9	50.6	47.6
	EA		54.4	54.1	**55.0**	52.4	48.2
	CAOTE		54.2	54.0	54.3	52.8	49.3
	Ours		**54.5**	**54.6**	54.8	**53.6**	**51.4**
MultiFieldQA-zh	SnapKV	42.5	40.8	40.4	39.2	36.5	31.4
	PyramidKV		41.4	40.8	39.9	37.3	32.7
	KVzip		41.1	40.9	40.4	38.4	35.2
	EA		41.5	42.1	**42.7**	40.2	35.8
	CAOTE		41.8	41.3	42.3	40.5	37.3
	Ours		**41.9**	**42.3**	42.5	**41.3**	**38.9**
Multi-document QA							
HotpotQA	SnapKV	58.8	57.2	56.5	54.6	48.6	40.0
	PyramidKV		57.4	56.7	54.5	49.8	42.5
	KVzip		57.8	57.2	55.7	51.7	46.1
	EA		57.7	58.1	58.7	55.5	48.9
	CAOTE		57.9	57.6	58.5	55.8	50.0
	Ours		**58.4**	**58.7**	**58.8**	**56.8**	**53.8**
2WikiMQA	SnapKV	54.5	52.6	52.2	50.5	44.6	35.7
	PyramidKV		53.1	52.7	50.7	45.8	38.3
	KVzip		53.1	52.8	51.5	47.8	41.9
	EA		53.9	54.0	**54.7**	51.1	44.8
	CAOTE		53.7	53.5	54.4	51.3	45.9
	Ours		**54.1**	**54.4**	54.4	**52.2**	**49.8**
MuSiQue	SnapKV	38.0	36.5	35.8	34.1	28.2	19.7
	PyramidKV		36.6	35.8	33.9	29.0	21.5
	KVzip		37.0	36.2	34.7	31.2	25.3
	EA		37.0	37.0	**38.4**	34.3	27.9

Continued on next page

Task	Method	Full	$\rho=0.10$	$\rho=0.25$	$\rho=0.50$	$\rho=0.75$	$\rho=0.90$
	CAOTE		37.2	37.1	37.6	35.3	29.4
	Ours		**37.3**	**38.0**	38.0	**36.4**	**33.0**
DuReader	SnapKV	36.5	35.0	34.3	32.5	26.4	18.2
	PyramidKV		35.2	34.3	32.6	27.7	20.3
	KVzip		35.6	35.1	33.2	29.6	24.1
	EA		35.3	35.8	**36.9**	33.3	26.8
	CAOTE		35.2	35.7	36.4	33.6	27.9
	Ours		**35.7**	**36.5**	36.3	**34.5**	**31.6**
Summarization							
GovReport	SnapKV	33.2	32.1	31.8	30.9	28.3	24.1
	PyramidKV		31.9	31.6	30.8	28.7	24.9
	KVzip		32.3	31.9	31.2	29.6	27.3
	EA		32.3	32.7	**33.2**	30.9	28.1
	CAOTE		32.0	32.5	32.4	31.0	28.7
	Ours		**32.6**	**32.9**	33.0	**31.6**	**30.7**
QMSum	SnapKV	26.0	24.9	24.4	23.5	21.3	17.3
	PyramidKV		25.1	24.2	23.6	21.4	17.8
	KVzip		25.1	24.5	24.5	22.7	20.3
	EA		25.0	25.6	**25.8**	24.2	21.3
	CAOTE		25.1	25.4	25.5	24.0	21.7
	Ours		**25.2**	**25.8**	**25.8**	**24.6**	**23.3**
MultiNews	SnapKV	28.4	27.3	26.8	26.2	23.4	19.7
	PyramidKV		27.1	26.7	26.1	24.1	20.6
	KVzip		27.4	26.9	26.5	24.7	22.5
	EA		27.8	27.6	**28.2**	26.4	23.2
	CAOTE		27.5	27.2	27.7	26.6	23.6
	Ours		**27.9**	**27.8**	28.0	**27.0**	**25.3**
VCSUM	SnapKV	17.2	16.1	15.5	14.7	12.2	8.4
	PyramidKV		16.0	15.8	14.9	12.7	9.1
	KVzip		16.0	16.0	15.4	13.8	11.5
	EA		16.3	16.5	**16.8**	15.0	12.3
	CAOTE		16.3	16.2	16.6	15.2	12.7
	Ours		**16.4**	**16.7**	16.7	**15.7**	**14.5**
Few-shot Learning							
TREC	SnapKV	74.2	72.8	72.2	70.3	65.0	56.4
	PyramidKV		72.6	72.5	70.7	66.3	58.7
	KVzip		73.1	72.6	71.3	67.4	61.6
	EA		73.4	73.6	**74.4**	70.3	64.7
	CAOTE		73.3	73.1	73.7	71.0	65.9
	Ours		**73.7**	**74.1**	74.2	**72.0**	**69.2**
TriviaQA	SnapKV	89.2	87.9	86.8	84.9	79.9	71.4
	PyramidKV		87.5	87.1	86.0	81.4	73.9
	KVzip		88.1	87.8	85.9	82.4	76.6
	EA		88.5	88.5	**89.6**	85.1	80.0
	CAOTE		87.9	88.3	88.9	86.1	81.4
	Ours		**88.8**	**89.2**	89.3	**86.2**	**84.2**
SAMSum	SnapKV	45.5	44.0	43.6	41.6	36.4	27.6
	PyramidKV		44.1	43.3	42.1	37.3	29.7
	KVzip		44.0	44.0	42.6	38.3	32.9
	EA		44.7	44.7	**45.5**	41.8	35.9
	CAOTE		44.3	44.7	45.3	41.9	37.7
	Ours		**44.9**	**45.5**	45.4	**43.8**	**40.9**
LSHT	SnapKV	33.2	31.5	31.1	28.9	24.1	15.2
	PyramidKV		32.0	31.3	30.0	25.5	18.0
	KVzip		32.1	31.8	30.2	26.0	20.9
	EA		32.4	32.4	**33.0**	29.2	23.9
	CAOTE		32.3	32.1	**33.0**	29.8	25.0
	Ours		**32.5**	**33.2**	32.9	**30.8**	**28.5**

Continued on next page

Task	Method	Full	$\rho=0.10$	$\rho=0.25$	$\rho=0.50$	$\rho=0.75$	$\rho=0.90$
Synthetic Tasks							
PassageCount	SnapKV	7.8	6.5	6.2	5.6	4.0	1.6
	PyramidKV		6.6	6.3	5.7	4.3	2.0
	KVzip		6.6	6.4	6.0	4.9	3.4
	EA		6.7	6.5	6.3	5.3	3.8
	CAOTE		6.6	6.5	6.3	5.5	4.0
	Ours		**6.8**	**6.7**	**6.5**	**6.0**	**5.6**
PassageRetrieval-en	SnapKV	91.0	89.5	88.0	83.8	67.0	39.5
	PyramidKV		89.0	87.5	83.0	69.5	44.5
	KVzip		89.7	88.6	86.5	74.5	56.5
	EA		90.1	90.4	**90.8**	81.0	62.5
	CAOTE		89.8	89.5	88.5	79.0	60.0
	Ours		**90.3**	**90.7**	90.5	**84.5**	**74.0**
PassageRetrieval-zh	SnapKV	76.5	74.8	72.5	67.5	53.0	30.5
	PyramidKV		74.5	72.8	68.5	55.5	34.0
	KVzip		74.9	73.5	71.0	61.0	45.0
	EA		75.5	75.9	**76.5**	66.0	49.0
	CAOTE		75.1	74.5	73.2	64.0	47.0
	Ours		**75.8**	**76.1**	76.2	**68.5**	**60.0**
Code Completion							
LCC	SnapKV	63.0	61.5	60.9	58.4	53.2	40.7
	PyramidKV		61.4	61.1	58.9	54.0	44.1
	KVzip		61.8	61.5	60.3	56.5	50.3
	EA		**62.5**	62.1	59.6	55.0	45.9
	CAOTE		61.9	62.0	60.3	56.5	49.8
	Ours		62.2	**62.6**	**61.8**	**59.0**	**55.2**
RepoBench-P	SnapKV	57.5	55.7	55.0	52.9	47.5	35.3
	PyramidKV		56.0	55.6	53.4	48.8	38.8
	KVzip		56.3	**57.2**	54.7	51.0	45.0
	EA		**56.7**	56.4	54.0	49.5	40.4
	CAOTE		56.4	56.4	54.9	51.1	43.7
	Ours		56.6	57.0	**57.0**	**54.0**	**50.2**
Overall Average							
Avg.	SnapKV	47.3	45.8	45.2	43.4	38.2	29.5
	PyramidKV		45.8	45.3	43.7	39.2	31.6
	KVzip		46.1	45.8	44.5	40.9	35.4
	EA		46.4	46.5	46.9	43.1	36.9
	CAOTE		46.3	46.2	46.4	43.4	37.8
	Ours		**46.6**	**47.0**	**47.0**	**44.8**	**41.8**

Table 5: Task-level LongBench results under prefill-time KV cache eviction on LLaMA3.1-8B. We report all 21 LongBench tasks, grouped by the six LongBench categories. Here, ρ denotes the fraction of evicted KV entries. Within each task block, **bold** marks the best score per column and underline marks the second-best. The Avg. row reports the arithmetic mean over all 21 tasks. EA denotes Expected Attention.

Aggregate prefill-time compression results

Model	Metric	Method	Full	$\rho=0.10$	$\rho=0.25$	$\rho=0.50$	$\rho=0.75$	$\rho=0.90$
Qwen3-8B	RULER 4K	SnapKV	94.7	92.0	84.5	76.2	66.7	54.9
		PyramidKV		91.5	85.0	75.5	69.3	52.7
		KVzip		93.7	92.2	86.8	74.0	67.4
		EA		94.2	93.8	92.4	84.5	63.1
		CAOTE		94.0	93.6	91.2	85.8	63.6
		Ours		**94.6**	**94.3**	**93.6**	**89.0**	**71.7**
	RULER 8K	SnapKV	86.1	85.4	84.5	78.8	62.7	47.9
		PyramidKV		85.6	84.3	80.6	64.8	49.3
		KVzip		85.7	85.0	81.8	66.2	52.4
		EA		85.5	85.2	80.7	69.4	54.0
		CAOTE		85.8	84.9	82.3	70.1	56.7
		Ours		**86.0**	**85.6**	**83.0**	**75.4**	**66.2**
	RULER 16K	SnapKV	79.5	78.7	77.4	70.8	54.9	37.1
		PyramidKV		79.0	77.7	72.5	56.4	39.6
		KVzip		78.9	77.8	73.4	60.4	45.2
		EA		79.1	77.6	72.6	59.4	44.2
		CAOTE		79.0	77.9	73.2	61.2	46.7
		Ours		**79.3**	**78.0**	**73.5**	**68.8**	**56.7**
LLaMA3.1-8B	RULER 4K	SnapKV	94.8	94.0	88.0	81.5	67.6	55.6
		PyramidKV		93.8	88.6	80.5	66.0	54.3
		KVzip		94.1	92.0	88.0	76.0	60.3
		EA		94.3	93.8	91.0	74.5	55.1
		CAOTE		94.2	93.2	89.8	77.0	60.0
		Ours		**94.7**	**94.4**	**93.0**	**81.0**	**68.8**
	RULER 8K	SnapKV	80.2	79.3	78.0	71.5	56.8	41.2
		PyramidKV		79.6	78.7	73.9	58.1	44.0
		KVzip		79.8	78.6	74.3	61.2	46.8
		EA		79.7	79.1	73.5	63.7	49.6
		CAOTE		79.9	78.9	75.2	63.1	51.0
		Ours		**80.0**	**79.5**	**76.0**	**73.8**	**61.2**
	RULER 16K	SnapKV	75.0	74.0	72.3	64.1	45.8	28.8
		PyramidKV		74.5	72.9	66.2	47.1	31.6
		KVzip		74.4	73.1	66.9	50.4	33.8
		EA		74.6	72.7	67.5	51.0	36.2
		CAOTE		74.3	73.6	68.0	56.1	40.6
		Ours		**74.8**	**73.8**	**70.5**	**66.3**	**52.5**
Gemma3-12B	RULER 4K	SnapKV	94.1	88.5	78.0	68.2	53.3	44.6
		PyramidKV		90.0	82.5	69.3	54.7	43.1
		KVzip		93.0	90.5	85.0	69.0	48.3
		EA		93.4	92.8	90.5	77.0	54.2
		CAOTE		93.6	93.2	91.2	78.5	55.5
		Ours		**94.0**	**93.8**	**92.6**	**81.0**	**61.6**
	RULER 8K	SnapKV	88.0	87.3	86.0	79.2	63.8	45.8
		PyramidKV		87.5	86.6	81.3	65.4	47.9
		KVzip		87.4	86.8	81.0	68.2	49.7
		EA		87.7	86.5	82.2	69.6	52.4
		CAOTE		87.6	87.0	82.9	71.0	55.8
		Ours		**87.8**	**87.2**	**83.8**	**76.0**	**68.0**
	RULER 16K	SnapKV	80.0	79.2	77.8	71.4	55.6	38.7
		PyramidKV		79.5	78.4	72.9	56.5	42.6
		KVzip		79.4	78.6	74.2	63.1	49.2
		EA		79.6	78.2	73.2	62.6	47.2
		CAOTE		79.3	78.8	74.6	62.8	49.6
		Ours		**79.8**	**79.0**	**75.5**	**68.0**	**60.0**

Table 6: We report prefill-time compression results on RULER across three models and context lengths of 4K, 8K, and 16K. Here, ρ denotes the fraction of evicted KV entries, and Full denotes the uncompressed KV cache. **Bold** and underline mark the best and second-best scores within each column, respectively.

E Efficiency Analysis

Table 7 reports the end-to-end quality–efficiency trade-off of cache compression, including task score, compression time, end-to-end latency, allocated GPU memory, and retained KV budget. Comp. measures eviction scoring, cache selection, and compressed-cache construction, excluding the initial prompt prefill; E2E includes prompt prefill, compression, and decoding.

For prefill-time compression, the prompt prefill is identical across methods and is not accelerated by cache eviction. We therefore evaluate on RULER 8K and report Decode Mem., the peak allocated GPU memory during answer generation after the compressed prompt cache has been materialized. This should be interpreted as post-prefill decoding-memory reduction, not as a reduction in the peak memory needed to ingest the original prompt. For generation-time compression, we evaluate on MATH500 with a 4096-token maximum generation length and report Online Mem., the peak allocated GPU memory during online generation with pruning.

Memory is measured as total allocated GPU memory, including model parameters and runtime buffers. At matched retained-KV budgets, our method preserves substantially more task quality at a higher compression-time cost. Cross-budget comparisons in Table 7 further show that it retains 2.5–3× fewer KV entries than SnapKV while improving task scores by 1.2–4.4 points. This increases E2E latency by 0.24–0.75 seconds relative to SnapKV, but remains faster than full-cache inference.

(a) Prefill-time compression on RULER 8K.

Model	Method	ρ	Score (%)	Δ Score (pts.)	Comp. (s/ex.)	E2E (s/ex.)	E2E Speedup (×)	Decode Mem. (GB)	Δ Mem. (GB)	KV Retained (%)
LLaMA3.1-8B	Full cache	–	80.2	0.0	–	8.91	1.00×	21.82	–	100%
	SnapKV	0.75	56.8	-23.4	0.24	7.18	1.24×	21.09	-0.73	25%
	CAOTE	0.75	63.1	-17.1	1.15	7.95	1.12×	21.20	-0.62	25%
	Ours	0.75	73.8	-6.4	1.68	8.11	1.10×	21.22	-0.60	25%
	SnapKV	0.90	41.2	-39.0	0.29	6.73	1.32×	20.88	-0.94	10%
	CAOTE	0.90	51.0	-29.2	1.32	7.49	1.19×	21.01	-0.81	10%
	Ours	0.90	61.2	-19.0	1.97	7.86	1.13×	21.03	-0.79	10%
Gemma3-12B	Full cache	–	88.0	0.0	–	12.58	1.00×	31.36	–	100%
	SnapKV	0.75	63.8	-24.2	0.41	10.08	1.25×	30.18	-1.18	25%
	CAOTE	0.75	71.0	-17.0	1.82	11.25	1.12×	30.34	-1.02	25%
	Ours	0.75	76.0	-12.0	2.27	11.64	1.08×	30.40	-0.96	25%
	SnapKV	0.90	45.8	-42.2	0.52	9.48	1.33×	29.82	-1.54	10%
	CAOTE	0.90	55.8	-32.2	2.04	10.58	1.19×	30.00	-1.36	10%
	Ours	0.90	68.0	-20.0	2.61	10.83	1.16×	30.07	-1.29	10%

(b) Generation-time compression on MATH500.

Model	Method	Budget	Score (%)	Δ Score (pts.)	Comp. (s/ex.)	E2E (s/ex.)	E2E Speedup (×)	Len. (tokens)	Online Mem. (GB)	Δ Mem. (GB)	Gen. KV Retained (%)
DeepSeek-R1 Qwen-7B	Full cache	–	84.0	0.0	–	15.12	1.00×	3870	23.58	–	100%
	SnapKV	4×	78.4	-5.6	0.29	13.55	1.12×	3825	23.20	-0.38	25.0%
	R-KV	4×	82.2	-1.8	0.96	14.05	1.08×	3927	23.29	-0.29	25.0%
	Ours	4×	82.8	-1.2	1.25	14.39	1.05×	3886	23.34	-0.24	25.0%
	SnapKV	12×	60.2	-23.8	0.43	12.87	1.17×	3746	22.91	-0.67	8.3%
	R-KV	12×	78.6	-5.4	1.19	13.49	1.12×	3861	23.02	-0.56	8.3%
	Ours	12×	79.8	-4.2	1.36	13.82	1.09×	3849	23.09	-0.49	8.3%
Qwen2.5 Math-7B	Full cache	–	82.5	0.0	–	13.68	1.00×	3565	22.13	–	100%
	SnapKV	4×	76.8	-5.7	0.22	12.35	1.11×	3504	21.82	-0.31	25.0%
	R-KV	4×	80.6	-1.9	0.76	12.86	1.06×	3622	21.91	-0.22	25.0%
	Ours	4×	81.2	-1.3	0.97	13.16	1.04×	3587	21.96	-0.17	25.0%
	SnapKV	12×	56.8	-25.7	0.31	11.74	1.17×	3441	21.58	-0.55	8.3%
	R-KV	12×	76.5	-6.0	0.88	12.25	1.12×	3529	21.68	-0.45	8.3%
	Ours	12×	78.0	-4.5	1.16	12.59	1.09×	3576	21.75	-0.38	8.3%

Table 7: Quality–efficiency comparison on a single A100 80GB GPU. Prefill results use RULER 8K, and generation-time results use MATH500 with a 4096-token generation limit. Δ Score is relative to full-cache decoding. Comp. measures cache-compression time, while E2E includes prompt prefill, compression, and decoding. Decode/Online Mem. reports peak allocated GPU memory, including model parameters and runtime buffers. KV budgets are matched within each compression setting.

Table 8 reports KV-cache-only memory and compression-time scaling across context lengths, model sizes, and batch sizes.

(a) Context scaling: LLaMA3.1-8B, batch size 1.

Context	Retained prompt-KV memory (GB)	Peak allocated memory (GB)	Compression time (s/ex.)
4K	0.13	20.70	1.02
8K	0.27	21.22	1.68
16K	0.54	22.25	2.95

(b) Model scaling: 8K context, batch size 1.

Model	Layers	Retained prompt-KV memory (GB)	Compression time (s/ex.)
LLaMA3.2-3B	28	0.20	0.96
LLaMA3.1-8B	32	0.27	1.68
Gemma3-12B	48	0.37	2.27

(c) Batch scaling: LLaMA3.1-8B, RULER 8K.

Batch size	Compression time per batch (s)	Compression time per example (s)
1	1.68	1.68
2	3.18	1.59
4	6.04	1.51

Table 8: KV-specific memory and compression-time scaling on a single A100 80GB GPU. All compressed settings use $\rho = 0.75$. Peak allocated memory includes model parameters and runtime buffers, whereas retained prompt-KV memory isolates the materialized cache after compression.

F Ablation Study

Exact optimization over all eviction masks is combinatorial. Table 9 evaluates the approximation introduced by restricting refinement to a candidate pool.

Selection method	RULER 8K		MATH500	
	Score	Time (s/ex.)	Score	Time (s/ex.)
Singleton top-B	71.9	1.22	78.2	1.02
Pool-refine	73.8	1.68	79.8	1.36
All-candidate refinement	74.1	6.10	80.2	4.90

Table 9: Approximation quality and compression cost of mask selection. RULER uses LLaMA3.1-8B at $\rho = 0.75$; MATH500 uses DeepSeek-R1-Distill-Qwen-7B under $12\times$ compression. Pool-refine recovers 86% and 80% of the score improvement from all-candidate refinement while being approximately $3.6\times$ faster.

Setting and model	$m = 1/4/8$	$K = 128/256/512$	Selection configurations
Prefill: LLaMA3.1-8B	70.4/**73.8**/73.5	**73.8**/74.0/73.9	72.8/73.2/73.6/**73.8**
Prefill: Gemma3-12B	73.9/**76.0**/75.8	**76.0**/76.1/76.0	75.0/75.4/75.8/**76.0**
Generation: DeepSeek-R1-Distill-Qwen-7B	78.8/**79.8**/79.5	79.2/**79.8**/79.9	78.9/79.3/**79.8**/79.7
Generation: Qwen2.5-Math-7B	77.0/**78.0**/77.8	77.4/**78.0**/78.1	77.2/77.5/**78.0**/77.9

Table 10: Cross-model hyperparameter sensitivity. Prefill results use RULER 8K at $\rho = 0.75$, and generation results use MATH500 under $12\times$ compression. Selection configurations, from left to right, are $(\alpha, b) = (1.5, 256), (2.0, 128), (2.0, 256), (2.0, 512)$. **Bold** marks the setting-specific default, which remains within 0.2 points of the best configuration in every case.

Ablation	Variant	Prefill-time compression		
		LongBench	RULER 8K	RULER 16K
Final-norm correction	w/o final-norm propagation	43.9	71.8	63.1
	with final-norm propagation	44.8	73.8	66.3
Scoring window	$m_{\text{pre}} = 1$	44.6	70.4	61.9
	$m_{\text{pre}} = 4$	44.8	73.8	66.3
	$m_{\text{pre}} = 8$	45.7	73.5	66.5
Top-K KL support	$K_{\text{pre}} = 128$	44.8	73.8	66.3
	$K_{\text{pre}} = 256$	44.9	74.0	66.2
	$K_{\text{pre}} = 512$	44.7	73.9	66.6
Selection refinement	Singleton	42.8	71.9	63.7
	Pool-refine, $\alpha = 1.5, b_{\text{pre}} = 256$	44.2	72.8	65.1
	Pool-refine, $\alpha = 2.0, b_{\text{pre}} = 128$	44.4	73.2	65.6
	Pool-refine, $\alpha = 2.0, b_{\text{pre}} = 256$	44.6	73.6	66.0
	Pool-refine, $\alpha = 2.0, b_{\text{pre}} = 512$	44.8	73.8	66.3

Table 11: Prefill-time ablation study of the main scoring and selection hyperparameters. Unless otherwise specified, all other hyperparameters are fixed to the prefill-time defaults in Table 3. Results are reported on LLaMA3.1-8B at $\rho = 0.75$. The default configuration uses $m_{\text{pre}} = 4$, $K_{\text{pre}} = 128$, $\alpha = 2.0$, and $b_{\text{pre}} = 512$.

Ablation	Variant	Generation-time compression	
		MATH500	AIME25
Final-norm correction	w/o final-norm propagation	78.5	15.6
	with final-norm propagation	79.8	17.8
Scoring window	$m_{\text{gen}} = 1$	78.8	15.6
	$m_{\text{gen}} = 4$	79.8	17.8
	$m_{\text{gen}} = 8$	79.5	16.7
Top-K KL support	$K_{\text{gen}} = 128$	79.2	16.7
	$K_{\text{gen}} = 256$	79.8	17.8
	$K_{\text{gen}} = 512$	79.9	17.8
Selection refinement	Singleton	78.2	13.3
	Pool-refine, $\alpha = 1.5, b_{\text{gen}} = 256$	78.9	15.6
	Pool-refine, $\alpha = 2.0, b_{\text{gen}} = 128$	79.3	16.7
	Pool-refine, $\alpha = 2.0, b_{\text{gen}} = 256$	79.8	17.8
	Pool-refine, $\alpha = 2.0, b_{\text{gen}} = 512$	79.7	17.8

Table 12: Generation-time ablation study of the main scoring and selection hyperparameters. Unless otherwise specified, all other hyperparameters are fixed to the generation-time defaults in Table 3. Results are reported on DeepSeek-R1-Distill-Qwen-7B under $12\times$ generation-cache compression with maximum generation length 4096. The default configuration uses $m_{\text{gen}} = 4$, $K_{\text{gen}} = 256$, $\alpha = 2.0$, and $b_{\text{gen}} = 256$.

G Artifact licenses and intended use.

We use publicly available models, datasets, benchmarks, and baseline implementations for research evaluation under their respective licenses and terms of use. Our use is consistent with their intended research and evaluation purposes. We report aggregate results only and do not redistribute original model checkpoints, datasets, or third-party code.

H Disclosure of the AI assistant

In accordance with the ACL AI writing assistance policy, we disclose the use of AI in this paper. We used ChatGPT, an AI language model developed by OpenAI, solely as a writing aid to enhance the clarity and readability of our manuscript.