SpectralKV: Redundancy-Aware KV Cache Compression via Spectral Coreset Selection

Fangming Zhao[1,†], **Xiaofei Yue**[2,†], **Fulun Ye**[1], **Yu Peng**[1],
Junyu Chen[1], **Tingting Li**[1], **Ziming Zhao**[1,*]

[1]School of Software Technology, Zhejiang University
[2]School of Computer Science and Technology, Beijing Institute of Technology

[†]Equal contribution [*]Corresponding author
zhaoziming@zju.edu.cn

Abstract

Long-context inference with large language models is limited by the key-value (KV) cache, which grows linearly with the input and is read at every decoding step. Most compression methods evict tokens: they score each cached token on its own, keep the highest-scoring ones, and delete the rest. We revisit this recipe from the perspective of the attention output and show that, when evicted tokens can be merged into kept ones, the output error decomposes exactly over the evicted tokens. The decomposition reveals that an evicted token can be recovered by a kept token whose attention is proportional to its own, so the benefit of keeping a token depends on whether other kept tokens already represent it. Building on this analysis, we propose **SpectralKV**, a training-free method that compresses the prompt KV cache into a weighted coreset. SpectralKV finds candidate representatives in a key-value space shaped by the covariance spectrum of the observed queries and by the output projection, and scores how much of each token's contribution they recover. It then selects the representatives that maximize importance-weighted coverage jointly across attention heads and merges the remaining tokens into them while preserving their attention mass. Across 29 datasets from LongBench and RULER on Llama-3.1-8B-Instruct and Qwen3-4B, SpectralKV retains 98.1% of the full-cache performance with 5% and 10% of the prompt cache and outperforms the strongest of six recent eviction methods by 3.2 points on average. Code and data are available at `https://github.com/Secbrain/SpectralKV`.

1 Introduction

Large language models (LLMs) are increasingly applied to long inputs, such as long documents, retrieved passages, code repositories, and multi-turn agent trajectories [1–4]. To avoid recomputing past states, autoregressive decoding stores the keys and values of all previous tokens in a key-value (KV) cache [5, 6]. The cache grows linearly with the input length: for Llama-3.1-8B in BF16, a single 128K-token prompt occupies 16 GiB, comparable to the model weights. Since the entire cache is read at every decoding step, reducing it is central to efficient long-context inference [7].

A widely used approach is KV cache eviction [8–10], which exploits the sparsity of attention: a small fraction of the cached tokens receives most of the attention [8]. Many methods therefore compress the prompt cache once after prefill, the forward pass over the prompt: they score each token by the attention it receives from the last few queries of the prompt, called the observation window [10], keep the highest-scoring tokens in each attention head, and delete the rest. Later work refines the score [11–15] and the split of the budget across layers and heads [16–18], and merging methods blend

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

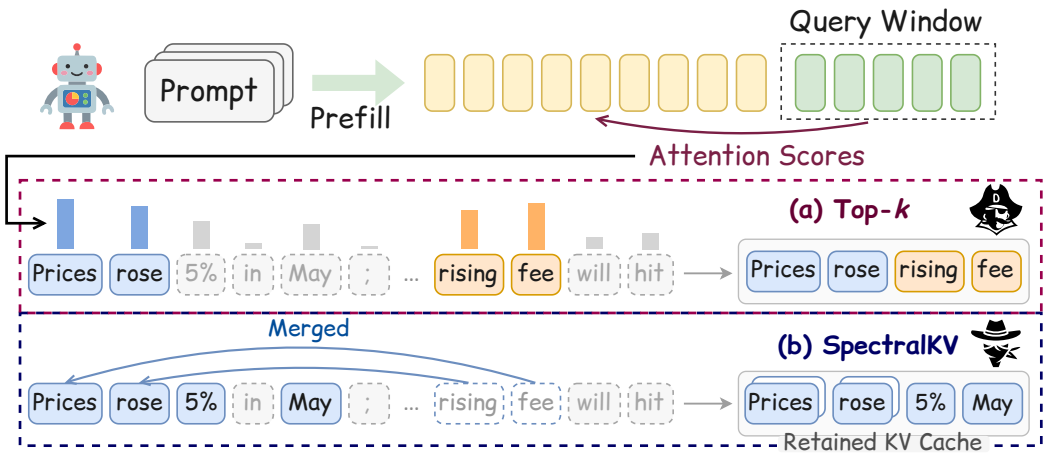


Figure 1: Top-k eviction (a) and SpectralKV (b) on one attention head with a budget of four tokens. Dashed tokens are not stored, and a stacked slot also carries the tokens merged into it. Top-k keeps *rising* and *fee* next to *rose* and *Prices*; SpectralKV merges them into the latter and keeps *5%* and *May*.

evicted tokens into kept ones to retain part of their information [19–21]. In all of them, however, the tokens to keep are chosen by ranking each token on its own (Figure 1a).

Ranking tokens on their own overlooks how the cache is used. Decoding uses the cached tokens only through the attention output, an average of their values weighted by attention. Two tokens can therefore be largely interchangeable: if the attention on one is nearly proportional to the attention on the other and their values are similar, keeping one of them and scaling up its attention reproduces nearly the combined contribution of both. Long inputs restate the same content many times, in the same or different words [22], and contain many such tokens, yet ranking cannot exploit them: interchangeable tokens all score high, so top-k selection keeps several of them where one with scaled attention would suffice, and evicted tokens are deleted when a kept token could stand in for them.

We make this view precise by treating the compressed cache as a weighted coreset: a small subset of tokens, each carrying a weight that scales its attention, such that attention over the subset approximates attention over the full cache. Eviction is the special case in which every weight equals one, and an evicted token can instead be merged into a kept token, its representative, by increasing the representative's weight and blending the evicted value into its value. The error of the attention output then decomposes exactly over the evicted tokens (Section 3): a deleted token loses its contribution no matter which other tokens are kept, whereas a token merged into a representative with proportional attention incurs no error. Once tokens can be merged, the benefit of keeping a token therefore depends on whether other kept tokens already represent it, not only on its own importance.

Based on this analysis, we propose **SpectralKV**, a training-free method that compresses the prompt KV cache into a weighted coreset (Figure 1b). SpectralKV estimates from the observation window how well tokens represent one another, by testing whether their attention is proportional and their values are close. It then selects representatives that maximize importance-weighted coverage, the total importance of the tokens they represent, and merges each remaining token into its best representative, preserving its attention mass. Joint selection across the heads of a layer gives each head the slots it needs, and greedy optimization guarantees a $(1 - 1/e)$ approximation.

We evaluate SpectralKV on Llama-3.1-8B-Instruct and Qwen3-4B [23, 24] over 29 datasets from LongBench [1] and RULER [25]. Keeping 5% and 10% of the prompt cache, SpectralKV retains 98.1% of the full-cache performance and outperforms six recent eviction methods in every setting, by 3.2 points on average over the strongest. A variant with a fixed layer profile, which compresses each layer during prefill, performs on par. The gains are largest where content repeats: in common-word extraction with Llama-3.1 at 5%, SpectralKV scores 70.0, against 32.4 for the best baseline.

In summary, this paper makes the following contributions:

- We formulate KV cache compression as building a weighted coreset and derive an exact error decomposition, which explains why deletion cannot exploit redundancy while merging can.

- We propose SpectralKV, which selects representatives by importance-weighted coverage with a $(1 - 1/e)$ guarantee and merges the rest into them while preserving their attention mass.
- On two models, 29 datasets, and two budgets, SpectralKV consistently outperforms six eviction methods, and ablations show that coverage-based selection and merging reinforce each other.

2 Related Work

KV Cache Eviction. Eviction methods keep a subset of the cached tokens and discard the rest. StreamingLLM [9] keeps a few initial tokens that act as attention sinks together with a recent window, and H2O [8], Scissorhands [26], and TOVA [27] keep tokens that receive high attention during decoding. SnapKV [10] compresses the prompt cache once after prefill using an observation window at the end of the prompt, which is the setting we adopt. Later methods refine the importance score by accounting for the output projection of values [11], aggregating scores robustly against their fluctuation [12], reconstructing layer outputs [13], anticipating the attention of future queries [14, 15], or measuring how well the compressed cache lets the model reconstruct the context [28]. All of these methods rank tokens individually and delete those that are not selected. Quantization [29, 30], low-rank projection [31], and sparse attention over the full cache [32, 33] complement eviction.

Budget Allocation. Another line of work decides how many tokens each layer or head keeps. PyramidKV [16] and PyramidInfer [34] keep fewer tokens in higher layers, CAKE [17] derives layer budgets from how attention is dispersed over tokens and how it shifts over time, Ada-KV [18] and HeadKV [35] distribute the budget across heads, and DuoAttention [36] keeps the full cache only for retrieval heads. SpectralKV also sets layer budgets by the variability of attention, but its head budgets follow from a single selection objective instead of a separate rule.

KV Cache Merging. Merging methods retain part of the information of evicted tokens. CaM [19] merges the values of tokens to be evicted into retained ones, D2O [20] merges evicted tokens into their most similar retained tokens, KVMerger [21] merges consecutive tokens with similar keys, and RazorAttention [37] summarizes the dropped tokens of each head into a compensation token. LESS [38] augments eviction with a learned recurrent state, and DMC [39] learns dynamic memory compression through continued pre-training. The training-free methods among them use importance, position, or similarity to decide which tokens to retain or combine, whereas SpectralKV merges a token only into a kept token whose attention is proportional to its own, preserves the merged attention mass through a logit bias, and uses the same relation to decide which tokens to keep.

Coreset Selection. A coreset is a small weighted subset that approximates a larger set with respect to a given objective [40]. A related matrix approximation problem is column subset selection, which selects original columns to approximate the full matrix [41]. Facility-location objectives, which reward a subset for how well it covers every element of the full set, are monotone and submodular, so greedy selection achieves a $(1 - 1/e)$ approximation [42]. They have been used to select sentences for summarization [43] and examples for efficient training [44]. SpectralKV treats the compressed cache as a weighted coreset of the attention over the prompt and uses a facility-location objective whose directed similarity measures how much of one token's contribution another token recovers.

3 Preliminaries and Analysis

3.1 Attention and Weighted Cache

We consider an LLM with L layers, each using grouped-query attention [45] with H KV heads that are each shared by G query heads of dimension d. We describe a single KV head and drop layer and head indices when no confusion arises. After the prefill of an N-token prompt, the head caches keys $k_i \in \mathbb{R}^d$ and values $v_i \in \mathbb{R}^d$ for $i = 1, \ldots, N$. A query q of query head g attends to them with weights $a_i(q)$ and produces the output $y(q)$,

$$a_i(q) = \frac{\exp\big(q^\top k_i/\sqrt{d}\big)}{\sum_l \exp\big(q^\top k_l/\sqrt{d}\big)}, \qquad y(q) = \sum_i a_i(q)\, v_i, \tag{1}$$

which the head adds to the residual stream as $W_O^{(g)} y(q)$, where $W_O^{(g)} \in \mathbb{R}^{D\times d}$ is the block of the output projection that belongs to query head g and D is the hidden size. Following SnapKV [10], the

last w prompt tokens are always kept, and their queries form the observation window, which serves as a proxy for the queries of future decoding steps. The other prompt tokens form the compressible set $\mathcal{C}$; each KV head keeps a subset $S \subseteq \mathcal{C}$ within a keep ratio p and evicts $\mathcal{E} = \mathcal{C} \setminus S$.

Eviction stores the kept tokens unchanged. We consider a weighted cache in which each kept token j stores its key k_j, a value $\tilde{v}_j$, and a mass $m_j \geq 1$ that multiplies its unnormalized attention, implemented as a bias $\log m_j$ on its attention logit. Such a cache is a weighted coreset of the full cache, and eviction is the special case $m_j = 1$ and $\tilde{v}_j = v_j$. An evicted token i can be merged into a kept token $\pi(i)$, its representative, with a ratio $r_i \geq 0$:

$$m_j = 1 + \sum_{i:\, \pi(i)=j} r_i, \qquad \tilde{v}_j = \frac{1}{m_j}\Big(v_j + \sum_{i:\, \pi(i)=j} r_i\, v_i\Big), \tag{2}$$

so that, besides its own attention, j receives r_i times that attention on behalf of each merged token i and contributes it with the value v_i. Deletion is the special case $r_i = 0$.

3.2 Decomposition of the Output Error

Let $\tilde{y}(q)$ be the attention output under the weighted cache, and define the attention mismatch of an evicted token $i \in \mathcal{E}$ as $\delta_i(q) = r_i\, a_{\pi(i)}(q) - a_i(q)$, the attention that its representative carries on its behalf minus the attention that it receives in the full cache.

Proposition 1 (Exact error decomposition)**.** *For every query q,*

$$\tilde{y}(q) - y(q) = \frac{\sum_{i\in\mathcal{E}} \delta_i(q)\,\big(v_i - y(q)\big)}{1 + \sum_{i\in\mathcal{E}} \delta_i(q)}, \tag{3}$$

where the denominator is the ratio of the total unnormalized attention under the weighted cache to that under the full cache and is therefore always positive.

Proposition 1, proved in Appendix A, makes two properties of compression explicit. First, a deleted token has $\delta_i(q) = -a_i(q)$, and its term in the numerator, $-a_i(q)\,(v_i - y(q))$, involves only the token itself: the cost of deleting a token does not depend on which other tokens are kept, and none of its contribution survives. Under deletion, a kept token cannot compensate for a similar deleted one, so redundancy among tokens cannot be exploited, and ranking tokens individually by their attention and the norm of their values after the output projection is the natural strategy [10, 11]. Second, a merged token causes no error on any query for which $a_i(q) = r_i\, a_{\pi(i)}(q)$, that is, when its attention is a scaled copy of the attention on its representative. With merging, the cost of evicting a token thus depends on whether a kept token represents it, which is a property of the kept set rather than of the token alone. A compression method should therefore measure how well one token represents another, choose a kept set that represents as much important content as possible without spending slots on represented tokens, and merge the rest into their representatives (Section 4).

4 SpectralKV

SpectralKV compresses the prompt cache of each layer after prefill (Figure 2 and Algorithm 1): it measures how well tokens represent one another (Section 4.1), selects the kept set by importance-weighted coverage jointly over the heads of the layer (Section 4.2), and merges the evicted tokens into their representatives (Section 4.3), under the layer budgets of Section 4.4. Since each token is compared with only a few candidates, the cost grows linearly with the prompt length.

4.1 Representation in a Spectral Key-Value Space

Representation Score. By Proposition 1, merging i into j is exact when $a_i(q) = r_i\, a_j(q)$ for every future query q, and we test this condition on the observation window. We index the Gw observation queries of a KV head by t, write $a_{i,t}$ for the attention of query t on token i, and split the queries into two halves by alternating positions, $\mathcal{T}_{\mathrm{fit}}$ and $\mathcal{T}_{\mathrm{val}}$. The merge ratio is fitted on the first half by least squares and capped by the ratio of total attention, so that the representative never carries more attention mass for i than i received, and the fraction of the attention on i recovered by the scaled copy

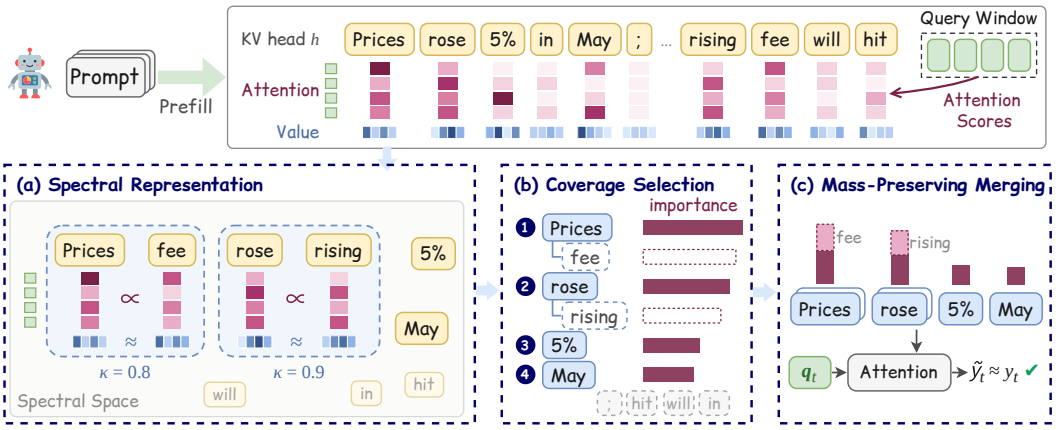


Figure 2: Overview of SpectralKV on one KV head, illustrated with the sentence of Figure 1. The top row shows the cache after prefill, with one row of attention per observation query and the value of each token. Panels (a) to (c) correspond to Sections 4.1 to 4.3: $\propto$ and $\approx$ mark proportional attention and close values, numbers give the greedy order, dashed bars mark covered tokens, and stacked bars show the attention mass that a kept token carries for merged ones.

is measured on the second half:

$$r(i \leftarrow j) = \min\left\{ \frac{\sum_{t\in\mathcal{T}_{\text{fit}}} a_{i,t}\, a_{j,t}}{\sum_{t\in\mathcal{T}_{\text{fit}}} a_{j,t}^2}, \frac{\sum_{t\in\mathcal{T}_{\text{fit}}} a_{i,t}}{\sum_{t\in\mathcal{T}_{\text{fit}}} a_{j,t}} \right\}, \tag{4}$$

$$\rho(i \leftarrow j) = \max\left\{ 0,\ 1 - \frac{\sum_{t\in\mathcal{T}_{\text{val}}} \left| r(i \leftarrow j)\, a_{j,t} - a_{i,t} \right|}{\sum_{t\in\mathcal{T}_{\text{val}}} a_{i,t}} \right\}. \tag{5}$$

Since proportionality on the window may fail for later queries, which then read the merged value of j, we also require the two values to be close after the output projection. We map values to $o_i = P_V^\top v_i$ with $P_V P_V^\top = \frac{1}{G}\sum_g W_O^{(g)\top} W_O^{(g)}$, so that $\|o_i - o_j\|^2$ is the mean squared difference between the contributions of v_i and v_j to the residual stream. Letting $\bar{o}$ be the average output of the observation window in this space, we define $\chi(i \leftarrow j) = \max\{0,\ 1 - \|o_i - o_j\|/\|o_i - \bar{o}\|\}$, which compares the value difference with $v_i - y$, the scale at which i enters Eq. (3), and the representation score

$$\kappa(i \leftarrow j) = \min\left\{\rho(i \leftarrow j),\ \chi(i \leftarrow j)\right\} \in [0, 1], \tag{6}$$

which estimates the fraction of the contribution of i that j can recover. The score is directed, since j may represent i without i representing j, and we set $\kappa(i \leftarrow i) = 1$.

Spectral Candidate Search. Evaluating κ for all pairs would take quadratic time, so we evaluate it only for candidate pairs that are likely to score high. Since $a_i(q)$ and $a_j(q)$ share the softmax normalizer, their ratio $\exp\big(q^\top(k_i - k_j)/\sqrt{d}\big)$ is constant across queries exactly when $q^\top(k_i - k_j)$ does not vary with q. For queries with covariance Σ, the variance of this logit difference is $(k_i - k_j)^\top \Sigma\,(k_i - k_j)/d$. A key difference therefore matters only along the directions in which queries vary, weighted by the corresponding eigenvalues of Σ, whereas a difference along a direction of constant query component only rescales the ratio, which r absorbs. We estimate Σ for each KV head from the observation queries of its G query heads, also rotated to later positions to anticipate future queries (Appendix C), and map the keys to $z_i = P_K^\top k_i$ with $P_K P_K^\top = \Sigma/d$, so that $\|z_i - z_j\|$ equals the standard deviation of the logit difference. Proportional pairs are thus close in this spectral key space, just as pairs with a high χ are close in the value space. We combine both into $x_i = [\, z_i;\ o_i/\sigma\,]$, where σ^2 is the mean of $\|o_i - \bar{o}\|^2$ over the tokens, take the eight nearest neighbors of each compressible token in this space, among the tokens of the same head within 2,048 positions, as its candidate representatives, and set $\kappa(i \leftarrow j) = 0$ for all other pairs.

4.2 Coverage-Based Coreset Selection

Coverage Objective. Let n_ℓ be the number of compressible slots of layer ℓ summed over its KV heads (Section 4.4), and let $c_{h,i} \geq 0$ be the importance of token i in KV head h, defined below.

Algorithm 1: SpectralKV prompt compression

Input: prompt KV cache of all layers, observation queries, keep ratio p
Output: weighted cache of kept keys, merged values, and log-masses

for $\ell \leftarrow 1$ **to** L **do** ▷ statistics during prefill
  $c \leftarrow$ importance weights of all heads ▷ Eq. (8)
  $\eta_\ell \leftarrow$ variability of the observation attention ▷ Eq. (9)
  $\mathcal{N} \leftarrow$ nearest neighbors in the spectral key-value space ▷ Section 4.1
  $r, \kappa \leftarrow$ merge ratios and representation scores on $\mathcal{N}$ ▷ Eqs. (4) to (6)
$n_1, \ldots, n_L \leftarrow$ divide the slots allowed by p by η_ℓ or θ_ℓ ▷ Section 4.4
for $\ell \leftarrow 1$ **to** L **do** ▷ compression
  $S_1, \ldots, S_H \leftarrow$ lazy greedy maximization of F with n_ℓ slots ▷ Eq. (7)
  foreach evicted token i of head h **do** merge i into $\arg\max_{j \in S_h} \kappa_h(i \leftarrow j)$ if $\kappa_h > 0$ ▷ Eq. (2)
  foreach KV head h **do** delete instead if merging is worse on $\mathcal{T}_{\text{val}}$

SpectralKV selects the kept sets $S_1, \ldots, S_H \subseteq \mathcal{C}$ of all heads jointly by solving

$$\max_{S_1,\ldots,S_H} F(S) = \sum_{h=1}^{H} \sum_{i \in \mathcal{C}} c_{h,i} \max_{j \in S_h} \kappa_h(i \leftarrow j) \quad \text{s.t.} \quad \sum_{h=1}^{H} |S_h| = n_\ell, \tag{7}$$

where the maximum over an empty set is zero. A token contributes to F either by being kept, since $\kappa(i \leftarrow i) = 1$, or through the kept token that represents it best. The objective therefore rewards tokens that are important on their own or that represent many important tokens, while a token already represented by a selected one adds little, so slots are not spent on copies. Because the heads of a layer share one budget, a head with many distinct important tokens receives more slots than a head whose important content is concentrated or repetitive. When no token represents another, F is the total importance of the kept tokens and Eq. (7) keeps the n_ℓ most important tokens across the heads of the layer, which is the head-adaptive top-k selection of Ada-KV [18].

Importance Weights. The importance should reflect how strongly the attention output depends on a token, which by Proposition 1 is governed by its attention and its value after the output projection [11]. We first score token i in KV head h by

$$p_{h,i} = \frac{\max_{g,t} \tilde{a}_{h,g,t,i}\, \nu_{h,g,i}}{\sum_{h'} \bar{\nu}_{h',i}}, \tag{8}$$

where $\nu_{h,g,i} = \|W_O^{(g)} v_{h,i}\|$ is the norm of the projected value, $\bar{\nu}_{h,i}$ its mean over the G query heads, and $\tilde{a}$ the observation attention smoothed along the token axis by a mean filter of width five that preserves the total attention, which favors the neighbors of attended tokens as pooling does in SnapKV [10]. The maximum over query heads and observation queries retains a token that any query reads strongly [28], and the denominator places all heads of the layer on a common scale, since they compete for one budget. Because a word split into subword tokens, such as a number, is useful only when its pieces are kept together, we finally propagate importance forward within each word, $c_{h,i} = \max_{k \le i,\, k \sim i} \gamma^{i-k} p_{h,k}$ with $\gamma = 0.95$, where $k \sim i$ denotes tokens of the same word.

Greedy Optimization. Eq. (7) is a weighted facility-location problem. Since $c \ge 0$ and $\kappa \in [0, 1]$ are fixed before selection, F is normalized, monotone, and submodular, so the greedy algorithm that repeatedly adds the token with the largest marginal gain attains at least a $(1 - 1/e)$ fraction of the optimum [42]. We run it with lazy evaluation on the sparse candidate graph, which yields the same selection as the plain greedy algorithm (Appendix C).

4.3 Mass-Preserving Merging

Representative Assignment. After selection, every evicted token i is merged into the kept candidate j with the largest $\kappa(i \leftarrow j) > 0$, using $r(i \leftarrow j)$ in Eq. (2), and tokens without such a representative are deleted. The representative keeps its key and position, so its own attention is unchanged, and its mass restores the attention mass of the tokens merged into it. The cache stores $\log m_j$ in FP32 next to the kept keys and merged values, the attention kernel adds it to the logit of j during decoding, and generated tokens are appended with a log-mass of zero; the log-masses take less than 1% of the memory of a BF16 key-value pair and are counted in the budget.

Merge Verification. Proposition 1 also shows that merged tokens interact through the shared normalizer, so merges that are accurate individually can jointly increase the error. We therefore verify merging for every KV head on the held-out queries $\mathcal{T}_{\mathrm{val}}$. With the merged values rounded to the storage precision, a head keeps the merged cache only if its projected output error $\sum_g \sum_{t \in \mathcal{T}_{\mathrm{val}}} \|W_O^{(g)}(\tilde{y} - y)\|^2$ is lower than with deletion of the same tokens.

4.4 Layer Budget Allocation

Variability-Based Allocation. The keep ratio fixes the size of the compressed cache, including log-masses and metadata, and thus the average number of rows B per KV head (Appendix B); besides the w window tokens of every head, the remaining $LH(B - w)$ slots are divided among layers. When the observation queries of a layer attend to the same tokens, a few kept tokens serve all of them, whereas when attention moves from query to query, future queries are likely to need a larger set. Following this intuition, which also underlies CAKE [17], we measure the variability of attention across the observation queries,

$$\eta_\ell = \frac{1}{HG} \sum_{h,g} \sum_{i \in \mathcal{C}} \mathrm{Var}_t\big(a^{(\ell)}_{h,g,t,i}\big), \tag{9}$$

and divide the slots in proportion to η_ℓ, capping each layer at its number of compressible tokens and rounding by the largest-remainder method so that the total is preserved. Layer ℓ then keeps b_ℓ rows per head and has $n_\ell = H(b_\ell - w)$ slots for the coverage objective in Eq. (7).

Fixed Layer Profile. Allocation by η_ℓ needs the statistics of all layers before any layer can be compressed, so we also consider a profile θ_ℓ calibrated offline for each model by averaging the layer shares of Eq. (9) over a small calibration set. All layer budgets are then known before prefill, so the two loops of Algorithm 1 merge into one that compresses each layer as soon as its attention has been computed, while head budgets and token selection still adapt to the input. We call this variant SpectralKV-Fixed; it performs on par with dynamic allocation (Section 5.5).

5 Experiments

We compare SpectralKV with six training-free eviction methods on LongBench and RULER (Sections 5.1 to 5.3), separate the contributions of its components (Section 5.4), and compare layer budget allocations and measure the compression overhead (Section 5.5).

5.1 Experimental Setup

Models and Benchmarks. We use Llama-3.1-8B-Instruct [23] and Qwen3-4B [24] in BF16, with the thinking mode of Qwen3 disabled and a fixed prompt template per model. From LongBench [1], we use the 16 English tasks with 100 fixed inputs per task; from RULER [25], we use all 13 tasks (eight needle-in-a-haystack variants, variable tracking, two aggregation tasks, and two QA tasks) at the 16K configuration with 50 inputs per task, for 2,250 inputs per model and budget. Scores use the official metric of each task on a 0 to 100 scale, with tasks weighted equally within each benchmark, and summaries over both benchmarks average their two averages (Appendix D.1). Ablations also report $\mathrm{KL}(p_{\mathrm{full}} \,\|\, p_{\mathrm{comp}})$ of next-token distributions over up to 32 output tokens.

Baselines. We compare with four methods that score tokens with the observation window, SnapKV [10], CriticalKV [11], Layer-DefensiveKV [12], and ReST-KV [13], and two that estimate future queries, Expected Attention [14] and DapQ [15]. Each keeps its own scoring and allocation, with the head-wise allocation of Ada-KV [18] for CriticalKV and Expected Attention (Appendix D.2).

Budgets and Decoding. We keep 5% and 10% of the KV cache of each tokenized prompt. For SpectralKV, the budget is counted in bytes and includes the log-masses and per-head metadata (Appendix B), so it stores about 1% fewer rows than the baselines. Compression is applied once after prefill, generated tokens are appended without eviction, and all methods decode greedily, taking the first output token from the uncompressed prefill. SpectralKV uses $w = 32$, a filter width of five,

Table 1: LongBench results at 5% and 10% keep ratios. SpectralKV uses dynamic layer budgets and SpectralKV-Fixed the fixed layer profile. Best and second-best compression results within each model and ratio block are shown in **bold** and underlined, respectively.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasp	MFQA	HpQA	2Wiki	Musi	GovR	QMS	MNws	TREC	TQA	SAM	PCnt	PsgR	LCC	Repo	
Llama-3.1-8B-Instruct (5% keep ratio)																	
Full Cache	24.53	43.69	57.06	54.07	42.78	33.06	33.81	25.56	27.11	76.00	90.89	43.15	8.20	100.00	60.54	55.75	48.51
SnapKV	23.51	29.16	55.31	53.08	40.66	**32.21**	23.05	24.43	18.32	59.00	89.32	41.68	8.14	**100.00**	55.51	54.54	44.25
CriticalKV	23.45	28.19	52.90	53.15	40.21	30.47	24.07	23.65	16.54	60.00	88.50	41.75	8.17	**100.00**	54.40	**56.61**	43.88
Layer-DefensiveKV	23.64	34.68	55.91	53.12	**43.41**	32.03	25.40	24.44	12.21	67.00	90.05	41.99	8.00	**100.00**	48.09	54.94	44.68
ReST-KV	**25.69**	31.64	56.07	**53.71**	41.57	31.53	24.93	24.40	19.05	70.00	90.82	43.69	8.00	**100.00**	55.94	54.36	45.71
Expected Attn.	22.14	25.74	36.39	42.10	30.56	23.88	22.28	21.80	**21.74**	31.00	74.56	36.61	8.00	97.42	35.90	34.19	35.27
DapQ	25.25	24.02	53.90	51.75	42.48	30.29	24.48	24.43	19.85	56.00	90.53	43.91	**8.25**	**100.00**	54.43	52.36	43.87
SpectralKV	25.30	37.34	56.36	53.67	42.04	32.05	**27.00**	**24.56**	20.95	72.00	89.96	**44.04**	8.00	**100.00**	57.53	55.99	46.68
SpectralKV-Fixed	24.04	**37.69**	**56.92**	53.53	41.78	32.08	26.83	24.42	21.04	**73.00**	**90.94**	43.11	8.00	**100.00**	**58.79**	55.41	**46.72**
Llama-3.1-8B-Instruct (10% keep ratio)																	
SnapKV	24.98	36.92	56.60	**54.00**	42.17	32.03	26.79	24.51	20.69	66.00	90.82	42.81	**8.25**	**100.00**	57.03	54.44	46.13
CriticalKV	24.09	37.71	55.57	53.09	41.90	31.59	26.84	23.81	21.92	69.00	90.42	41.49	**8.25**	**100.00**	58.71	55.11	46.22
Layer-DefensiveKV	25.19	41.30	**57.04**	53.92	41.63	32.36	**29.40**	24.48	21.90	**75.00**	90.39	42.33	8.00	**100.00**	57.50	**55.74**	47.26
ReST-KV	25.02	39.01	56.60	53.60	42.80	31.98	27.01	24.70	21.84	73.00	90.64	**43.66**	**8.25**	**100.00**	56.63	54.71	46.84
Expected Attn.	24.43	32.49	47.49	45.64	38.47	25.68	23.72	22.53	22.86	39.00	83.60	39.76	8.00	99.00	44.87	40.64	39.89
DapQ	25.16	31.58	55.55	52.76	**43.03**	32.14	26.36	**24.83**	21.53	61.00	90.10	43.19	8.20	**100.00**	55.25	53.51	45.26
SpectralKV	**25.35**	42.19	56.94	53.74	42.63	32.87	28.82	24.77	**23.53**	73.00	**90.94**	43.17	8.20	**100.00**	**59.02**	55.18	**47.52**
SpectralKV-Fixed	24.68	**43.24**	56.80	53.85	41.69	**32.95**	28.96	24.79	23.19	73.00	90.62	43.32	8.20	**100.00**	58.95	54.76	47.44
Qwen3-4B (5% keep ratio)																	
Full Cache	23.44	42.25	55.80	53.70	46.39	34.77	32.32	24.86	25.47	56.00	92.02	41.35	1.50	100.00	2.06	1.14	39.57
SnapKV	**24.42**	33.20	53.13	53.83	44.54	35.23	22.37	24.24	15.79	43.50	91.36	39.72	0.00	**100.00**	5.38	1.95	36.79
CriticalKV	23.13	32.60	49.59	53.49	43.83	35.52	21.96	23.93	14.16	49.50	**91.76**	40.05	0.50	**100.00**	**6.11**	2.62	36.80
Layer-DefensiveKV	23.62	36.13	53.02	53.37	45.40	**36.24**	23.36	24.11	17.30	50.00	91.36	40.89	0.50	**100.00**	4.56	1.53	37.59
ReST-KV	23.11	32.66	50.47	53.08	44.42	35.07	23.45	22.59	17.10	**56.50**	89.76	40.96	1.00	**100.00**	2.90	0.95	37.13
Expected Attn.	16.78	29.84	29.76	37.28	30.73	22.43	22.56	21.67	**20.38**	14.00	75.02	31.10	0.00	95.00	2.67	**4.36**	28.35
DapQ	24.19	32.05	49.99	51.90	42.48	34.07	22.88	24.47	17.50	40.00	**91.76**	**42.03**	0.50	**100.00**	2.86	1.76	36.15
SpectralKV	23.93	37.38	53.31	**53.88**	**45.45**	34.58	**25.70**	**24.98**	18.78	52.00	91.36	41.07	0.50	**100.00**	3.63	1.11	37.98
SpectralKV-Fixed	23.13	**37.92**	**53.76**	53.84	45.21	34.90	25.28	24.76	18.25	54.00	91.36	41.01	**1.50**	**100.00**	3.95	1.17	**38.13**
Qwen3-4B (10% keep ratio)																	
SnapKV	23.80	36.44	53.54	54.12	44.80	35.37	26.08	25.03	18.77	52.00	91.36	40.22	**0.50**	**100.00**	3.14	0.85	37.88
CriticalKV	**24.72**	35.52	53.04	53.33	45.03	34.82	26.06	24.32	19.16	52.00	**92.02**	40.20	**0.50**	**100.00**	**3.74**	2.34	37.93
Layer-DefensiveKV	23.91	38.90	55.41	53.47	**46.73**	**35.98**	26.08	24.95	20.02	55.00	**92.02**	40.84	**0.50**	**100.00**	3.30	1.96	38.69
ReST-KV	23.36	35.72	53.45	53.98	45.87	34.49	26.65	24.18	20.07	**56.50**	91.86	41.12	**0.50**	**100.00**	2.49	0.93	38.20
Expected Attn.	19.93	35.97	39.46	44.77	41.15	24.27	25.94	23.26	**22.23**	31.00	87.49	35.69	0.00	98.00	1.60	**2.45**	33.33
DapQ	23.66	36.45	52.76	53.68	45.89	35.45	25.24	**25.64**	19.46	46.00	91.59	**41.67**	**0.50**	**100.00**	3.31	2.10	37.71
SpectralKV	23.48	40.55	**55.75**	**54.17**	46.02	35.41	**28.38**	25.33	21.14	53.00	91.36	40.92	**0.50**	**100.00**	2.22	1.38	**38.72**
SpectralKV-Fixed	23.50	**40.68**	55.53	53.69	46.69	34.51	27.98	25.25	21.10	51.00	91.36	40.32	**0.50**	**100.00**	2.71	1.24	38.50

$\gamma = 0.95$, eight candidates within 2,048 positions, and dynamic layer budgets, and SpectralKV-Fixed uses the fixed layer profile of Section 4.4 instead, calibrated on 58 inputs per model (Appendix D.1).

5.2 Results on LongBench

Average Scores. SpectralKV outperforms all baselines on the LongBench average in all four settings (Table 1). The margin is largest at 5%, where it leads the strongest baseline by 0.96 points on Llama-3.1-8B-Instruct and by 0.39 points on Qwen3-4B and retains 96.2% and 96.0% of the full-cache average. SpectralKV-Fixed performs on par with SpectralKV and attains the highest average among all compression methods at 5% on both models.

Gains by Task. SpectralKV gains most on tasks over a single long document: it outperforms every baseline on Qasper in all four settings, by up to 2.66 points, and on MultiFieldQA-en and GovReport in three settings. Merging contributes most on summarization, where removing it lowers GovReport by 0.81 to 1.49 points in every setting and SAMSum by up to 2.51 points.

Table 2: RULER results. S3 and K3 are needle tasks with UUID values, and CWE and FWE are word extraction tasks. Avg. covers all 13 tasks (Table 6).

Method	*Llama-3.1-8B-Instruct*					*Qwen3-4B*				
	S3	K3	CWE	FWE	Avg.	S3	K3	CWE	FWE	Avg.
5% keep ratio										
Full Cache	100.00	100.00	83.40	88.00	92.24	100.00	98.00	71.60	96.00	91.20
SnapKV	22.00	8.00	32.40	88.00	73.98	48.00	16.00	28.60	98.67	74.00
CriticalKV	2.00	14.00	17.60	78.00	70.80	48.00	14.00	16.80	99.33	73.71
Layer-DefensiveKV	46.00	42.00	27.20	66.67	77.58	26.00	50.00	42.40	97.33	76.60
ReST-KV	**100.00**	70.00	17.20	**94.00**	85.32	98.00	22.00	17.20	**100.00**	80.58
Expected Attn.	0.00	0.00	2.60	3.33	20.03	0.00	0.00	6.00	46.00	36.08
DapQ	18.00	10.00	31.80	82.67	71.83	12.00	8.00	12.20	81.33	60.59
SpectralKV	**100.00**	**100.00**	**70.00**	88.67	**90.91**	**100.00**	82.00	**53.20**	98.00	88.55
SpectralKV-Fixed	**100.00**	98.00	68.20	90.00	90.78	**100.00**	**88.00**	50.00	98.00	**88.77**
10% keep ratio										
SnapKV	72.00	32.00	55.60	**93.33**	82.56	96.00	38.00	49.40	99.33	83.99
CriticalKV	12.00	54.00	41.00	83.33	77.77	86.00	38.00	36.40	98.67	81.77
Layer-DefensiveKV	98.00	98.00	60.40	72.00	88.97	90.00	92.00	55.60	98.67	88.28
ReST-KV	**100.00**	84.00	35.40	**93.33**	87.76	**100.00**	60.00	39.00	**100.00**	85.95
Expected Attn.	0.00	0.00	13.80	39.33	38.49	2.00	0.00	30.20	89.33	61.21
DapQ	66.00	26.00	50.60	86.00	80.45	56.00	18.00	25.20	98.00	74.70
SpectralKV	**100.00**	**100.00**	78.40	88.67	91.91	**100.00**	96.00	**65.60**	98.67	90.60
SpectralKV-Fixed	**100.00**	**100.00**	**78.80**	89.33	**91.99**	**100.00**	**98.00**	64.00	98.67	**90.63**

5.3 Results on RULER

Average Scores. SpectralKV and SpectralKV-Fixed attain the two highest RULER averages in all four settings (Table 2). At 5%, SpectralKV leads the strongest baseline, ReST-KV, by 5.60 points on Llama-3.1-8B-Instruct (90.91 vs. 85.32) and by 7.98 points on Qwen3-4B (88.55 vs. 80.58). At 10%, it leads the strongest baseline, Layer-DefensiveKV, by 2.94 and 2.32 points. Over both benchmarks, SpectralKV leads the strongest baseline, ReST-KV, by 3.2 points and retains 98.1% of the full-cache score on average over the four settings (Tables 1 and 2).

Repeated Content. The largest gains occur on common-word extraction (CWE), whose input is a long word list in which a few words recur many times and whose answer consists of the most frequent words. SpectralKV scores 70.0, 78.4, 53.2, and 65.6 in the four settings, against at most 32.4, 60.4, 42.4, and 55.6 for the baselines. Most of this gain comes from merging: without it, the same selection scores 13.2 to 24.8 points lower, since deleting the evicted copies of a word discards how often it occurs, whereas merging keeps their attention mass on the representative copy. On frequent-word extraction (FWE), SpectralKV matches or exceeds the full cache in every setting.

Long Values. On S3 and K3, whose needle values are UUIDs of many subword tokens, SpectralKV scores 100 and at least 82 in all settings, whereas at 5% the best baseline on K3 reaches 70 and 50 on the two models. The variant without merging scores similarly, so these gains come from the selection. On the remaining tasks (Table 6), SpectralKV stays within 4 points of the full cache.

5.4 Component Analysis

Coverage and Merging. Table 3a crosses how the kept set is selected with what happens to the evicted tokens, on a development set drawn from the evaluation set (Appendix D.1). Top-k selection replaces κ by the identity in Eq. (7), keeping the most important tokens across the heads of each layer, with all other components unchanged. Merging improves both selection rules, by 0.66 points with top-k and 1.00 point with coverage, and lowers the KL by 19% and 30%. Coverage, in contrast, pays off together with merging: with deletion, it changes the mean score by 0.07 points and the KL by 2%,

Table 3: Ablations. Scores average LongBench and RULER.

Variant	Llama		Qwen		Mean	KL
	5%	10%	5%	10%		
(a) Development set						
Top-k, delete	64.44	65.46	61.07	63.02	63.50	0.136
Top-k, merge	65.25	66.68	61.88	62.80	64.15	0.110
Coverage, delete	64.50	66.00	60.56	63.20	63.56	0.133
Coverage, merge	65.63	67.01	62.07	63.56	64.57	0.093
(b) Full evaluation set						
SpectralKV	68.79	69.71	63.27	64.66	66.61	0.103
w/o merging	67.80	68.90	62.71	64.04	65.86	0.135
fixed profile	68.75	69.71	63.45	64.57	66.62	0.104
uniform layers	68.66	69.66	62.93	64.40	66.41	0.109

Table 4: Prefill time and overhead.

Stage	Time (s)	
	Llama	Qwen
16K tokens, 5% ratio		
Prefill	2.64	2.00
Overhead, dynamic	0.74	0.78
Overhead, fixed	0.69	0.67
32K tokens, both ratios		
Prefill	6.42	5.37
Overhead, dynamic	1.47	1.73
Overhead, fixed	1.31	1.41

whereas with merging, it adds 0.41 points, wins in all four settings, and lowers the KL by another 16%. This interaction follows from Proposition 1: under deletion, a token left out because another token represents it loses its contribution all the same, so coverage exchanges it for a less important token, whereas with merging the representative recovers it. On the full evaluation set (Table 3b), removing merging lowers the mean score by 0.75 points (95% interval $[0.52, 0.97]$) and raises the KL by 32%; without merging, SpectralKV still outperforms every baseline.

5.5 Layer Budgets and Overhead

Layer Budget Allocation. The two variability-based allocations perform alike: SpectralKV-Fixed differs from SpectralKV by at most 0.22 points on each benchmark average and by at most 2 points on 114 of the 116 task scores (Tables 1 and 6), and on the full evaluation set it changes the mean score by $+0.01$ points (95% interval $[-0.11, 0.14]$) and raises the KL by 1.7% (Table 3b). Uniform layer budgets score lower than both in all four settings, by 0.20 and 0.21 points on average with 95% intervals that exclude zero, and raise the KL by 6.6%. Non-uniform budgets thus help mainly through the model-specific profile rather than through per-input adjustment (Appendix D.3), so SpectralKV-Fixed, which compresses each layer during prefill, is a natural choice for deployment [46].

Compression Overhead. On one H20 GPU, compression adds 0.74 and 0.78 s at about 16K tokens with dynamic budgets (Table 4), 28% and 39% of the prefill time, and SpectralKV-Fixed reduces it to 0.69 and 0.67 s by compressing each layer while later layers are still being prefilled. At about 32K tokens, building the caches of both ratios from one prefill adds 23% and 32%, since prefill attention grows quadratically whereas compression grows roughly linearly. The overhead is paid once per prompt, after which each decoding step reads a 10 to 20 times smaller cache.

6 Conclusion

We showed that, once evicted tokens can be merged, the benefit of keeping a token depends on whether other kept tokens represent it, and built SpectralKV to select representatives by importance-weighted coverage and merge the rest into them. On two models, 29 datasets, and two budgets, SpectralKV retains 98.1% of the full-cache performance and outperforms six recent eviction methods, with dynamic as well as fixed layer budgets. Extending the coverage objective and mass-preserving merging to compression during decoding is a natural next step (Appendix E).

Acknowledgments and Disclosure of Funding

This work was supported by the Ningbo Yongjiang Talent Programme (2025A-387-G), the Zhejiang Provincial Natural Science Foundation of China (Grant No. LQN26F020008), the CCF-Kuaishou Large Model Explorer Fund (No. CCF-KuaiShou 2025001), the CAAI-Ant Group Research Fund (2025CAAI-ANT-15), and the Science and Technology Innovation Project of Ningbo Global Innovation Center, Zhejiang University (Grant No. S02J2604).

References

[1] 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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137, 2024.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[4] Tingting Li, Ziming Zhao, Zhaoxuan Li, Xiaofei Yue, and Jiongchi Yu. Fair and Carbon-Aware LLM Routing for Web Services. In *Proceedings of the ACM Web Conference 2026*, pages 9563–9571, 2026.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 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*, pages 5998–6008, 2017.

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626, 2023.

[7] Xiaofei Yue, Song Yang, Liehuang Zhu, Stojan Trajanovski, and Xiaoming Fu. Demeter: Fine-grained function orchestration for geo-distributed serverless analytics. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 2498–2507, 2024.

[8] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *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*, 2023.

[9] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

[10] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM Knows What You are Looking for Before Generation. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[11] Yuan Feng, Junlin Lv, Haoyu Guo, Yukun Cao, S. Kevin Zhou, and Xike Xie. CriticalKV: Optimizing KV Cache Eviction from an Output Perturbation Perspective. In *Proceedings of the 43rd International Conference on Machine Learning, ICML 2026*, 2026.

[12] Yuan Feng, Haoyu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. DefensiveKV: Taming the Fragility of KV Cache Eviction in LLM Inference. In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*, 2026.

[13] Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV Cache Eviction with Layer-wise Output Reconstruction and Spatial-Temporal Smoothing. In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*, 2026.

[14] Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected Attention: KV Cache Compression by Estimating Attention from Future Queries Distribution. *CoRR*, abs/2510.00636, 2025.

[15] Zhenxu Tian, Yi Su, Juntao Li, and Min Zhang. Where Matters More Than What: Decoding-aligned KV Cache Compression via Position-aware Pseudo Queries. *CoRR*, abs/2603.11564, 2026.

[16] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. In *Second Conference on Language Modeling, COLM 2025*, 2025.

[17] Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. CAKE: Cascading and Adaptive KV Cache Eviction with Layer Preferences. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[18] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

[19] Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache Merging for Memory-efficient LLMs Inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pages 58840–58850, 2024.

[20] Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. D2O: Dynamic Discriminative Operations for Efficient Long-Context Inference of Large Language Models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[21] Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model Tells You Where to Merge: Adaptive KV Cache Merging for LLMs on Long-Context Tasks. *CoRR*, abs/2407.08454, 2024.

[22] Fangming Zhao, Xiaofei Yue, Fulun Ye, Yu Peng, Ziming Zhao, Lewei Jin, Wei Wu, Tianxiang Chen, Tingting Li, and Jianwei Yin. Semantic Retrieval Speculative Decoding with Intermediate Representations. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, pages 810–824, 2026.

[23] Llama Team. The Llama 3 Herd of Models. *CoRR*, abs/2407.21783, 2024.

[24] Qwen Team. Qwen3 Technical Report. *CoRR*, abs/2505.09388, 2025.

[25] 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*, 2024.

[26] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. In *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*, 2023.

[27] Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are Multi-State RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 18724–18741, 2024.

[28] Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

[29] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pages 32332–32344, 2024.

[30] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[31] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: KV-Cache Compression with Low-Rank Projection. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[32] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pages 47901–47911, 2024.

[33] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025.

[34] Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3258–3270, 2024.

[35] Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[36] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[37] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. RazorAttention: Efficient KV Cache Compression Through Retrieval Heads. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[38] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pages 11437–11452, 2024.

[39] Piotr Nawrot, Adrian Lancucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pages 37396–37412, 2024.

[40] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578, 2011.

[41] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 968–977, 2009.

[42] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.*, 14(1):265–294, 1978.

[43] Hui Lin and Jeff A. Bilmes. A Class of Submodular Functions for Document Summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 510–520, 2011.

[44] Baharan Mirzasoleiman, Jeff A. Bilmes, and Jure Leskovec. Coresets for Data-efficient Training of Machine Learning Models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 6950–6960, 2020.

[45] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901, 2023.

[46] Xiaofei Yue, Ziming Zhao, Jiongchi Yu, Huidong Ma, Zhaoxuan Li, Tingting Li, and Jianwei Yin. sMoE: Elastic MoE-Based Inference with Serverless Computing via Fine-grained Expert Scaling. In *ACM/IEEE Design Automation Conference*, 2026.

[47] Xiaofei Yue, Fangming Zhao, Fulun Ye, Jiongchi Yu, Zhaoxuan Li, Tingting Li, Ziming Zhao, and Jianwei Yin. Heterosim: Towards high-fidelity heterogeneous llm training simulation on gpus. In *Proceedings of the ACM Web Conference 2026*, pages 5189–5197, 2026.

[48] Xiaofei Yue, Song Yang, Fan Li, Youqi Li, and Yu Wang. Rocket: Warming serverless inference via hierarchical ML artifact pre-loading and sharing. In *IEEE INFOCOM 2026-IEEE Conference on Computer Communications*, pages 1–10, 2026.

A Proof of Proposition 1

Fix a query q and write $s_l = q^\top k_l/\sqrt{d}$ and $Z = \sum_l e^{s_l}$, where l ranges over all tokens of the full cache, so that $a_l = e^{s_l}/Z$. Let $\mathcal{K}$ denote the rows of the weighted cache, namely the kept tokens S, the observation window, and the generated tokens, so that every token of the full cache lies either in $\mathcal{K}$ or in $\mathcal{E}$. Each row $l \in \mathcal{K}$ has logit $s_l + \log m_l$ and value $\tilde{v}_l$, with $m_l = 1$ and $\tilde{v}_l = v_l$ for rows into which no token is merged. Dividing the numerator and the denominator of the compressed attention output by Z gives

$$\tilde{y} = \frac{\sum_{l\in\mathcal{K}} a_l\, m_l\, \tilde{v}_l}{\sum_{l\in\mathcal{K}} a_l\, m_l}.$$

By Eq. (2), $m_j \tilde{v}_j = v_j + \sum_{i:\pi(i)=j} r_i v_i$, so the numerator equals

$$\sum_{l\in\mathcal{K}} a_l v_l + \sum_{i\in\mathcal{E}} r_i\, a_{\pi(i)}\, v_i = y - \sum_{i\in\mathcal{E}} a_i v_i + \sum_{i\in\mathcal{E}} r_i\, a_{\pi(i)}\, v_i = y + \sum_{i\in\mathcal{E}} \delta_i v_i,$$

and, since $\sum_l a_l = 1$, the denominator equals

$$\sum_{l\in\mathcal{K}} a_l + \sum_{i\in\mathcal{E}} r_i\, a_{\pi(i)} = 1 - \sum_{i\in\mathcal{E}} a_i + \sum_{i\in\mathcal{E}} r_i\, a_{\pi(i)} = 1 + \sum_{i\in\mathcal{E}} \delta_i.$$

Hence $\tilde{y} = \big(y + \sum_{i\in\mathcal{E}} \delta_i v_i\big)\big/\big(1 + \sum_{i\in\mathcal{E}} \delta_i\big)$, and subtracting y yields Eq. (3). The denominator equals $\sum_{l\in\mathcal{K}} a_l m_l > 0$, which is the ratio of the total unnormalized attention under the weighted cache to that under the full cache, as stated in Proposition 1.

B Budget Accounting

Let $b_{\mathrm{KV}} = 2d \cdot s$ be the number of bytes of one key-value pair stored with s bytes per element ($s = 2$ for BF16). A keep ratio p allows $\lfloor pN \rfloor\, LH\, b_{\mathrm{KV}}$ bytes for the compressed prompt cache. Every kept row additionally stores a 4-byte log-mass, and every head stores 16 bytes of metadata for its offset and length. The average number of rows per KV head is therefore

$$B = \left\lfloor \frac{\lfloor pN \rfloor\, b_{\mathrm{KV}} - 16}{b_{\mathrm{KV}} + 4} \right\rfloor,$$

so that the total size $LHB\,(b_{\mathrm{KV}} + 4) + 16LH$ never exceeds the budget. The w window rows of every head are included in B, and a configuration with $B < w$ is reported as infeasible. Generated tokens are appended without eviction and are not part of the prompt budget.

C Implementation Details

Observation Statistics. The observation attention is computed with the queries of the last $w = 32$ prompt tokens at their original positions. Importance weights use the attention in the storage precision, while the representation score, the layer statistic η_ℓ, and the verification of merging use FP32. The query covariance is estimated from the observation queries of all query heads that share a KV head, each taken at its original position and additionally rotated by the rotary offsets $\tau \in \{8, 16, \ldots, 128\}$, which places it at later positions as a proxy for future decoding queries. The factor P_K is obtained by a Cholesky decomposition and, if the decomposition fails, by an eigendecomposition with negative eigenvalues clipped to zero; P_V depends only on the model weights and is computed once.

Candidate Graph. For efficiency, the tokens that search for candidates are restricted to blocks of 32 consecutive compressible positions that, summed over heads and ranked by importance, cover at least 99% of the total importance of the layer. For each such token, a shortlist of 16 nearest neighbors is retrieved within $\pm 2{,}048$ positions in the same head, and the 8 nearest are scored with Eq. (6). Distances are computed in BF16 with FP32 squared norms after scaling the features of each head by a positive constant, which does not change the order of distances.

Greedy Selection and Calibration. The greedy algorithm maintains marginal gains on the reverse adjacency of the sparse graph and evaluates them lazily, which returns the same order as the plain greedy algorithm. Since the greedy order does not depend on the budget, one run serves several keep ratios by taking prefixes of different lengths. The fixed layer profile θ_ℓ of each model is calibrated on 58 inputs, two from each task, by averaging the per-input layer shares with equal weight per task.

D Experimental Details

D.1 Evaluation Protocol

Inputs and Metrics. LongBench contributes 100 fixed inputs for each of its 16 English tasks, scored with the official metric of each task, and RULER contributes 50 inputs for each of its 13 tasks at the 16K configuration, scored by string matching. On Qwen3-4B, all methods, including the full cache, score below 7 on the two code tasks of LongBench. Scores are averaged within each task, then over tasks with equal weight within each benchmark; summaries over both benchmarks, such as the scores in Table 3, average the two benchmark averages, and averages over models and budgets weight each setting equally. For the KL divergence, we feed the greedy output of the full cache, up to 32 tokens after the first output token, to both caches and average $\mathrm{KL}(p_{\mathrm{full}} \,\|\, p_{\mathrm{comp}})$ of the next-token distributions over these positions; task and benchmark averages are then computed as for the scores.

Development Set and Calibration. The development set of Table 3a is a subset of the evaluation inputs that covers every task of both benchmarks for each model. The fixed layer profile is calibrated on 58 inputs per model, two per task, which do not overlap with the development set but belong to the evaluation set. Excluding them from the comparison in Table 3b changes the difference between the fixed and dynamic allocations from $+0.01$ to $+0.02$ points (95% interval $[-0.12, 0.15]$) and leaves the KL ratio of the fixed to the dynamic allocation unchanged at 1.017.

Statistical Comparisons. Confidence intervals come from a paired bootstrap with 2,000 resamples, stratified by model and task, in which both budgets of an input are resampled together and each resample is averaged in the same way as the reported scores.

Timing. The observation statistics, candidate search, and merging run on the GPU with fused kernels, and the greedy selection runs on the CPU and overlaps with GPU work on other layers [47]. Timings use one NVIDIA H20 GPU with BF16, SDPA attention, batch size one, and one model per GPU. After a warm-up run, each configuration is repeated five times, alternating with the plain prefill. The measured time spans from the start of prefill, including runtime initialization, until all compression work has finished, and excludes model loading, the first compilation, and decoding. The overhead is the median of this time minus the median time of a plain prefill of the same input.

Table 5: Configurations of the compared methods. Class names follow our implementations.

Method	Configuration
SnapKV	Observation window of 32 positions; average pooling with kernel size 5.
CriticalKV	`CriticalAdaKVPress` with a SnapKV scorer: observation window of 64 positions, kernel size 5, first-stage ratio 0.5; head-wise budgets of Ada-KV.
Layer-DefensiveKV	Observation window of 32 positions; kernel size 5; critical-mass parameter 0.9.
ReST-KV	Pooling kernel size 21 on RULER and 5 on LongBench.
Expected Attention	`AdaKVPress(ExpectedAttentionPress(`$\epsilon = 10^{-2}$`))`, i.e., combined with the head-wise budget allocation of Ada-KV.
DapQ	Pseudo window of 32 tokens, formed by the first 4 and the last 28 prompt tokens and assigned future positions; query-head scores are averaged within each GQA group; every layer and KV head keeps the same number of entries.
SpectralKV	Observation window of 32 positions; mean filter of width 5 that preserves the total attention; importance propagated within words with $\gamma = 0.95$; query covariance from the window queries, also rotated by the offsets $\{8, 16, \ldots, 128\}$; 8 scored candidates from a shortlist of 16 nearest neighbors within $\pm 2{,}048$ positions; lazy greedy selection jointly over the KV heads of each layer; merging verified per KV head on held-out window queries; dynamic layer budgets; budget counted in bytes, including log-masses and metadata (Appendix B).
SpectralKV-Fixed	As SpectralKV, but with the fixed layer profile of each model (Appendix C).

D.2 Baseline Configurations

Table 5 lists the configuration of every method as we ran it. Each baseline keeps its own settings rather than a common window or pooling width, so window and kernel sizes differ. Every baseline keeps $\lfloor pN \rfloor$ rows per KV head on average. For CriticalKV and Expected Attention, the head-wise allocation of Ada-KV is realized by masking evicted entries, so their scores correspond to the number of kept rows rather than to the physical memory they occupy.

D.3 Layer Allocation Analysis

For each input, we normalize the slots that the dynamic allocation assigns to each layer, excluding the window rows, into layer shares and compare them with the fixed profile by total variation, half the sum of absolute differences. The median distance is 5.0% on Llama-3.1-8B-Instruct and 5.2% on Qwen3-4B, with 90th percentiles of 7.1% and 7.2%, whereas the fixed profile is 11.5% and 18.9% away from uniform shares. Per-input adjustment thus moves the allocation much less than the model-specific profile does, which explains why the two allocations perform alike.

E Limitations

SpectralKV adds a one-time overhead to prefill (Table 4), which is amortized over decoding but matters for short generations [48]. Like other methods based on an observation window, it assumes that the queries at the end of the prompt anticipate those of future decoding steps: the representation score is fitted on these queries and merging is verified on held-out window queries, so merged tokens are not guaranteed to be accurate for every future query. The analysis and experiments cover a single compression after prefill; extending the coverage objective to repeated compression during decoding, where the cache grows with every generated token, is left to future work.

F Full RULER Results

Table 6 lists the scores of all methods on the 13 RULER tasks.

Table 6: Results on all 13 RULER tasks. S1 to S3 are single-needle tasks, K1 to K3 multi-key tasks, MV and MQ multi-value and multi-query tasks, VT variable tracking, CWE and FWE common-word and frequent-word extraction, and QA1 and QA2 question answering.

Method	S1	S2	S3	K1	K2	K3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg.
Llama-3.1-8B-Instruct (5% keep ratio)														
Full Cache	100.00	100.00	100.00	100.00	100.00	100.00	99.00	99.50	99.20	83.40	88.00	78.00	52.00	92.24
SnapKV	**100.00**	**100.00**	22.00	**100.00**	**100.00**	8.00	96.00	89.00	98.40	32.40	88.00	76.00	**52.00**	73.98
CriticalKV	**100.00**	98.00	2.00	**100.00**	**100.00**	14.00	91.50	92.50	98.80	17.60	78.00	76.00	**52.00**	70.80
Layer-DefensiveKV	**100.00**	98.00	46.00	**100.00**	**100.00**	42.00	**99.50**	**100.00**	99.20	27.20	66.67	**78.00**	**52.00**	77.58
ReST-KV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	70.00	**99.50**	**100.00**	98.40	17.20	**94.00**	**78.00**	**52.00**	85.32
Expected Attn.	92.00	2.00	0.00	4.00	26.00	0.00	4.50	1.50	44.40	2.60	3.33	52.00	28.00	20.03
DapQ	**100.00**	98.00	18.00	98.00	98.00	10.00	84.00	88.50	94.80	31.80	82.67	**78.00**	**52.00**	71.83
SpectralKV	**100.00**	96.00	**100.00**	**100.00**	**100.00**	**100.00**	**99.50**	98.50	99.20	**70.00**	88.67	**78.00**	**52.00**	**90.91**
SpectralKV-Fixed	**100.00**	96.00	**100.00**	**100.00**	**100.00**	98.00	99.00	99.00	**100.00**	68.20	90.00	**78.00**	**52.00**	90.78
Llama-3.1-8B-Instruct (10% keep ratio)														
SnapKV	**100.00**	98.00	72.00	**100.00**	**100.00**	32.00	98.50	97.00	98.80	55.60	**93.33**	76.00	**52.00**	82.56
CriticalKV	**100.00**	**100.00**	12.00	**100.00**	**100.00**	54.00	96.50	97.00	**99.20**	41.00	83.33	76.00	**52.00**	77.77
Layer-DefensiveKV	**100.00**	**100.00**	98.00	**100.00**	**100.00**	98.00	**99.50**	**99.50**	**99.20**	60.40	72.00	**78.00**	**52.00**	88.97
ReST-KV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	84.00	**99.50**	**99.50**	**99.20**	35.40	**93.33**	**78.00**	**52.00**	87.76
Expected Attn.	98.00	20.00	0.00	20.00	94.00	0.00	16.50	12.00	82.80	13.80	39.33	64.00	40.00	38.49
DapQ	**100.00**	**100.00**	66.00	**100.00**	**100.00**	26.00	93.50	95.00	98.80	50.60	86.00	**78.00**	**52.00**	80.45
SpectralKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.00	**99.50**	**99.20**	78.40	88.67	**78.00**	**52.00**	91.91
SpectralKV-Fixed	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**99.50**	99.00	**99.20**	**78.80**	89.33	**78.00**	**52.00**	**91.99**
Qwen3-4B (5% keep ratio)														
Full Cache	100.00	100.00	100.00	100.00	100.00	98.00	100.00	100.00	100.00	71.60	96.00	68.00	52.00	91.20
SnapKV	**100.00**	**100.00**	48.00	**100.00**	70.00	16.00	92.00	99.50	99.20	28.60	98.67	60.00	50.00	74.00
CriticalKV	**100.00**	**100.00**	48.00	**100.00**	66.00	14.00	98.50	**100.00**	97.60	16.80	99.33	64.00	**54.00**	73.71
Layer-DefensiveKV	**100.00**	**100.00**	26.00	**100.00**	88.00	50.00	90.00	98.50	93.60	42.40	97.33	60.00	50.00	76.60
ReST-KV	**100.00**	**100.00**	98.00	**100.00**	**100.00**	22.00	99.50	**100.00**	94.80	17.20	**100.00**	64.00	52.00	80.58
Expected Attn.	98.00	48.00	0.00	40.00	0.00	0.00	42.50	45.00	87.60	6.00	46.00	36.00	20.00	36.08
DapQ	**100.00**	**100.00**	12.00	**100.00**	32.00	8.00	53.00	93.50	95.60	12.20	81.33	58.00	42.00	60.59
SpectralKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	82.00	**100.00**	**100.00**	**100.00**	**53.20**	98.00	**66.00**	52.00	88.55
SpectralKV-Fixed	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**88.00**	**100.00**	**100.00**	**100.00**	50.00	98.00	**66.00**	52.00	**88.77**
Qwen3-4B (10% keep ratio)														
SnapKV	100.00	**100.00**	96.00	**100.00**	96.00	38.00	98.00	**100.00**	99.20	49.40	99.33	64.00	52.00	83.99
CriticalKV	100.00	**100.00**	86.00	**100.00**	88.00	38.00	**99.50**	**100.00**	98.40	36.40	98.67	64.00	**54.00**	81.77
Layer-DefensiveKV	100.00	**100.00**	90.00	**100.00**	**100.00**	92.00	**99.50**	99.50	98.40	55.60	98.67	64.00	50.00	88.28
ReST-KV	100.00	**100.00**	**100.00**	**100.00**	**100.00**	60.00	**99.50**	**100.00**	98.80	39.00	**100.00**	**66.00**	**54.00**	85.95
Expected Attn.	100.00	86.00	2.00	92.00	46.00	0.00	87.50	91.50	93.20	30.20	89.33	48.00	30.00	61.21
DapQ	100.00	98.00	56.00	**100.00**	74.00	18.00	89.50	**100.00**	98.40	25.20	98.00	**66.00**	48.00	74.70
SpectralKV	100.00	**100.00**	**100.00**	**100.00**	**100.00**	96.00	**99.50**	**100.00**	**100.00**	**65.60**	98.67	**66.00**	52.00	90.60
SpectralKV-Fixed	100.00	**100.00**	**100.00**	**100.00**	**100.00**	**98.00**	**99.50**	**100.00**	**100.00**	64.00	98.67	**66.00**	52.00	**90.63**

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes].

   Justification: The main claims are stated in the abstract and Section 1. They are supported by the analysis in Section 3, the method in Section 4, the experimental results in Section 5, and the limitations discussion in Appendix E.

2. **Limitations**

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

   Answer: [Yes].

Justification: Limitations are discussed in Appendix E, including the one-time prefill overhead, the reliance on the observation window, and the one-shot prefill compression setting.

3. **Theory assumptions and proofs**

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

   Answer: [Yes].

   Justification: Proposition 1 in Section 3.2 states its assumptions, and Appendix A gives its complete proof. The $(1 - 1/e)$ guarantee of the greedy selection in Section 4.2 follows from the cited result for monotone submodular maximization.

4. **Experimental result reproducibility**

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

   Answer: [Yes].

   Justification: Section 5.1 describes the models, benchmarks, decoding protocol, KV budget protocol, and baselines, and Algorithm 1 summarizes the method. Appendices B to D provide the budget accounting, implementation details, evaluation protocol, hardware, and baseline configurations.

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

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

   Answer: [Yes].

   Justification: The code and data repository is linked in the abstract. The experiments use publicly available models and benchmarks, with the evaluation protocol described in Appendix D.1.

6. **Experimental setting/details**

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

   Answer: [Yes].

   Justification: The experimental setup is described in Section 5.1, including models, prompt handling, baselines, hyperparameters, benchmarks, keep ratios, and decoding protocol. Additional details are provided in Appendices C and D.

7. **Experiment statistical significance**

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

   Answer: [Yes].

   Justification: Sections 5.4 and 5.5 report 95% confidence intervals for the component and layer-budget comparisons, computed with a paired bootstrap described in Appendix D.1. The main tables report task-level scores and averages without error bars: the evaluations use fixed benchmark samples and greedy decoding, and long-context inference is costly.

8. **Experiments compute resources**

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

   Answer: [Yes].

   Justification: Appendix D.1 reports the GPU type and the timing setup. Section 5.5 and Table 4 further report the prefill time and compression overhead.

9. **Code of ethics**

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

Answer: [Yes].

Justification: The work uses public models and benchmarks for inference-efficiency evaluation and does not involve private data, human subjects, or safety-critical deployment.

10. **Broader impacts**

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

    Answer: [N/A].

    Justification: The paper proposes a general KV cache compression method for efficient LLM inference and does not introduce a new application-facing model or dataset. Potential societal impacts are indirect and inherit from the underlying LLMs.

11. **Safeguards**

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

    Answer: [N/A].

    Justification: The paper does not release a new pretrained model, scraped dataset, or other asset with high direct misuse risk. The released asset is an inference-efficiency implementation.

12. **Licenses for existing assets**

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

    Answer: [No].

    Justification: The paper cites the existing models, datasets, and software used in the experiments, but the current version does not explicitly list all corresponding licenses and terms of use.

13. **New assets**

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

    Answer: [Yes].

    Justification: The paper releases code as a new asset through the repository linked in the abstract. No new dataset or model checkpoint is introduced.

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

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

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

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

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

    Answer: [N/A].

    Justification: The paper does not involve human-subject research, so IRB approval or equivalent review is not applicable.

16. **Declaration of LLM usage**

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

Answer: [N/A].

Justification: LLMs are used as evaluation backbones in standard inference experiments, not as an original or non-standard component of the research methodology.