Expected Contribution Matching: Optimal Values Merging for KV Cache Compression

Pengzhan Li **Yixuan Wang** **Wanxiang Che***
Research Center for Social Computing and Interactive Robotics,
Harbin Institute of Technology, China
{pzli,yixuanwang,car}@ir.hit.edu.cn

Abstract

Large Language Models (LLMs) use KV cache for efficient auto-regressive generation, but its linearly growing memory footprint bottlenecks long-context inference. Existing compression methods typically merge evicted information into retained tokens by additively combining both key and value vectors, despite their distinct roles in the attention mechanism. However, this uniform treatment **overlooks the distinct roles of keys and values in attention**: perturbations to keys are amplified through the Softmax exponential, whereas perturbations to values remain bounded and linear after Softmax. Based on this observation, we propose a differentiated strategy that evicts keys while merging values. Specifically, we formulate value merging as an MSE minimization problem and derive Expected Contribution Matching (ECM), a closed-form scheme that redistributes evicted values by their historical attention contributions. We further preserve semantic coherence by routing evicted information to semantically similar retained tokens via global semantic matching. Experiments on LongBench and LongBenchV2 demonstrate the effectiveness of our method, which achieves the highest overall averages among the evaluated training-free merging-based and eviction-based approaches under the same cache budget. Our code is available at https://github.com/Lpzhan931/ECM.

1 Introduction

Large Language Models (LLMs) are increasingly deployed in long-context applications where the KV Cache, which stores historical keys and values states to avoid redundant computation, becomes a critical memory bottleneck in batched and high-throughput inference (Kwon et al., 2023; Gao et al., 2025). Compressing the KV Cache without degrading model output is thus a pressing research challenge.

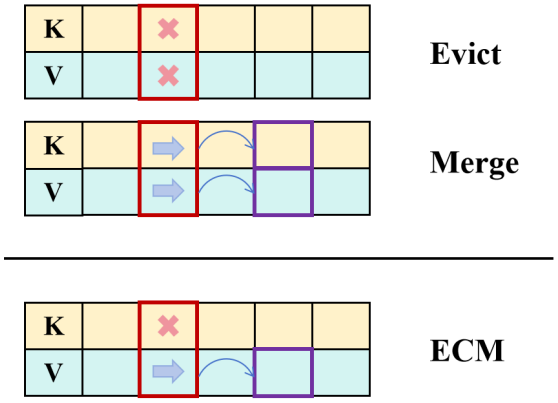


Figure 1: Illustration of different KV cache compression strategies. **Top:** Typical Evict and Merge methods. **Bottom:** Evicting keys while merging values (ECM).

KV Cache compression methods broadly fall into two categories: quantization and sparsity (Gao et al., 2025). Early sparsity work adopted selective eviction, retaining critical tokens such as sink tokens, recent windows (StreamingLLM; (Xiao et al., 2024)), and Heavy Hitters (H2O; (Zhang et al., 2023)). Their shared weakness is irreversibility: discarded tokens may carry information that cannot be recovered. A later line of work shifted to merging: information from evicted tokens is redistributed into retained KV pairs to reduce output perturbation. CaM (Zhang et al., 2024) uniformly distributes each evicted value across a local window. D2O (Wan et al., 2025) and AsymKV (Cui and Xu, 2025) merge both keys and values. KeepKV (Tian et al., 2026) introduces logarithmic correction aimed at achieving lossless merging.

Notably, most of these methods apply additive merging operation to both keys and values. We argue this overlooks a fundamental difference: keys and values play entirely different computational

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 19446–19458
October 24-29, 2026 ©2026 Association for Computational Linguistics

roles in attention. Keys participates in dot products with the Query (QK^T) inside the Softmax exponent, so any additive perturbation is exponentially transformed into a multiplicative perturbation on attention scores. Values participates in the post-Softmax weighted summation ($O = \sum a_i V_i$), a linear operation where errors remain bounded. Our analysis (Section 3.2) shows that additive keys merging introduces exponential multiplicative errors, while values merging produces only linear and bounded errors.

Building on this insight, we propose a differentiated strategy: merge values while evicting keys. For values merging, we address two sub-problems: which target values to merge into, and how to allocate the merging weights. For **target selection**, we select the top-K semantically similar retained values via global cosine-similarity search. For **weight allocation**, we formulate an MSE minimization problem and derive Expected Contribution Matching (ECM), a closed-form optimal solution that allocates weight proportionally to historical attention, provably superior to any alternative allocation under MSE. Together, these designs form a complete plug-and-play, training-free KV cache compression framework. We evaluate ECM on LongBench and LongBenchV2 using Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3, and compare it with both eviction-based and merging-based training-free baselines under a unified cache budget of 2048 tokens. ECM achieves the highest overall average among the evaluated compression methods in each main experimental setting. Controlled experiments further show that each component, including keys eviction, ECM weight allocation, and global semantic matching, contributes independently to the overall improvement. In addition, ECM introduces lower memory overhead than representative merging-based approaches such as AsymKV. Supplementary evaluations on Qwen2.5-14B-Instruct and RULER 4K are provided in Appendix B.

In summary, this paper makes the following contributions:

1. **Differentiated treatment of keys and values**: We provide the first systematic argument that keys vectors are unsuitable for additive merging, while values vectors are well-suited.

2. **The ECM merging algorithm**: We derive a closed-form optimal weight allocation for values merging and combine it with global semantic matching for target selection.

3. **Strong empirical results**: Our method achieves the highest overall averages among the evaluated training-free merging- and eviction-based methods on LongBench (Bai et al., 2024) and LongBenchV2 (Bai et al., 2025) under the same cache budget.

2 Related Work

KV Cache compression is a central problem in LLM inference optimization (Gao et al., 2025). We review related work from eviction-based to merging-based methods and highlight how each differs from our approach. In the process of literature survey, we used AI tools to improve efficiency.

2.1 Eviction-based Methods

Eviction-based methods retain critical tokens and discard the rest by designing importance metrics. StreamingLLM (Xiao et al., 2024) leverages the Attention Sink phenomenon, keeping initial sink tokens and a recent window. H2O (Zhang et al., 2023) dynamically retains Heavy Hitters based on cumulative attention scores. Other notable methods include FastGen (Ge et al., 2024) (adaptive per-layer eviction), ScissorHands (Liu et al., 2023) (one-time prefill-level eviction), Quest (Tang et al., 2024) (query-aware sparsity), SnapKV (Li et al., 2024) (prefix voting), and PyramidKV (Cai et al., 2024) (layer-wise pyramidal budgets). TRIM-KV (Bui et al., 2025) learns a retention-gating MLP, making it a learning-based method not directly comparable to our training-free approach. The shared limitation is irreversibility: evicted information cannot be recovered, which directly motivates merging-based methods.

2.2 Merging-based Methods

Merging-based methods compensate for eviction by merging discarded information into retained KV pairs. CaM (Zhang et al., 2024) uniformly distributing evicted values across a local consecutive window and using Bernoulli sampling to decide when to merge. It directly evicts keys without theoretical justification. D2O (Wan et al., 2025) selects targets by keys' cosine similarity and merges both keys and values. AsymKV (Cui and Xu, 2025) designs separate K/V schemes based on their different local properties, but still applies additive merging to keys through curvature-weighted averaging. KeepKV (Tian et al., 2026) introduces a logarithmic correction for lossless merging, yet when the attention

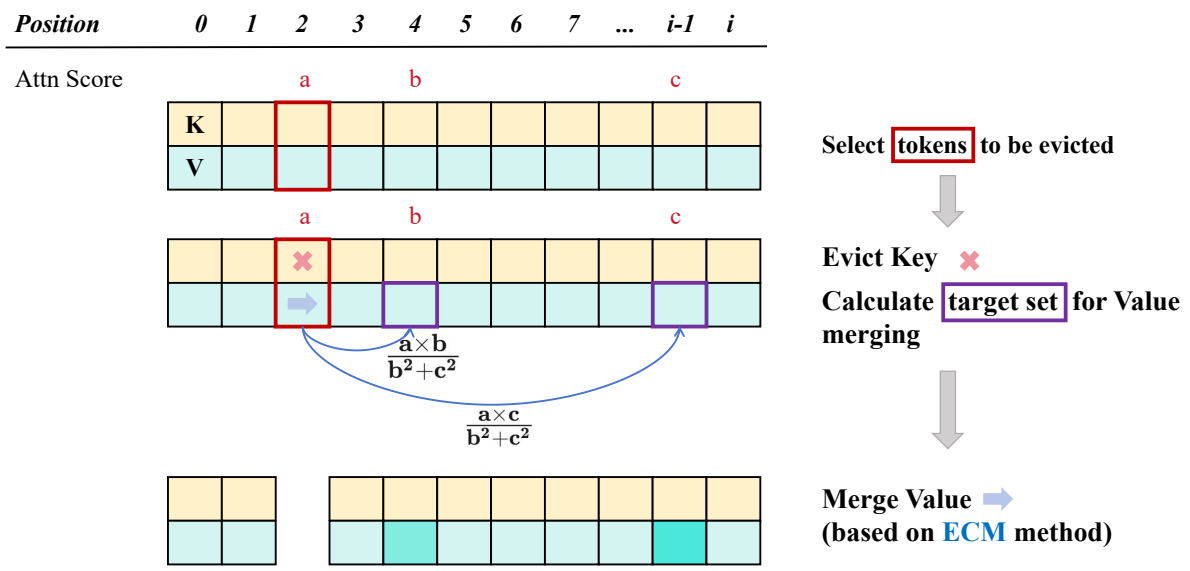


Figure 2: Overview of the ECM algorithm. Our proposed KV Cache compression algorithm consists of three stages. The red box identifies tokens to be evicted, chosen based on cumulative attention scores. Keys vectors are directly evicted to avoid the exponential multiplicative perturbation introduced by additive merging. The purple box marks the target set for values merging, selected globally based on values vector cosine similarity. The target set undergoes values merging using the ECM closed-form optimal formula, which allocates weight in proportion to historical attention scores.

consistency assumption is violated, the nonlinear coupling in its keys' formula causes complex error propagation. In most of these methods, the keys vector undergoes additive perturbation. Our work differs by directly evicting keys based on their structural unsuitability for merging, and by providing a provably optimal values merging scheme via ECM.

3 Methodology

3.1 Preliminary

Standard Attention. In autoregressive Transformers (Vaswani et al., 2017), each attention head computes the output $o_t = \sum_{j=1}^{N} a_{tj} v_j$ at step t, where the attention scores are:

$$a_{tj} = \frac{\exp(q_t \cdot k_j / \sqrt{d})}{\sum_{l=1}^{N} \exp(q_t \cdot k_l / \sqrt{d})} \tag{1}$$

Keys k_j and values v_j of all historical tokens are stored as the KV Cache to avoid recomputation, but its memory footprint grows with sequence length, becoming the bottleneck in long-context inference.

KV Cache Compression Objective. Compression reduces N cached KV pairs to $B < N$ pairs while minimizing output perturbation:

$$\min_{\mathcal{C}_{\text{comp}}} \mathbb{E}_q \left[\left\| \sum_{j=1}^{N} a_j v_j - \sum_{m=1}^{B} \tilde{a}_m \tilde{v}_m \right\|^2 \right] \tag{2}$$

where $\mathcal{C}_{\text{comp}} = \{(\tilde{k}_m, \tilde{v}_m)\}_{m=1}^{B}$ denotes the compressed cache.

Merging Framework. In merging-based compression, an evicted token i has its value distributed across a target set T of retained tokens: $\tilde{v}_t = v_t + w_{i \to t} \cdot v_i$ for $t \in T$. The compressed output decomposes as:

$$\begin{aligned} o_{\text{approx}} &= \sum_{t \in T} \tilde{a}_t \tilde{v}_t + \sum_{j \notin T \cup \{i\}} \tilde{a}_j v_j \\ &= \sum_{j \neq i} \tilde{a}_j v_j + v_i \underbrace{\sum_{t \in T} w_{i \to t} \tilde{a}_t}_{\tilde{a}_i} \end{aligned} \tag{3}$$

Comparing this with the true output $o_{\text{true}} = \sum_{j \neq i} a_j v_j + a_i v_i$, the core objective of merging strategy reduces to: using the attention scores of retained tokens to approximately reconstruct the attention score of the evicted token, i.e., $\tilde{a}_i = \sum_{t \in T} w_{i \to t} \tilde{a}_t \approx a_i$.

Attention Consistency Assumption. Since future Queries are unknown at compression time, we follow prior work in assuming historical cumulative attention $\hat{a}_j$ positively correlates with future attention:

$$a_j = c \cdot \hat{a}_j + \epsilon_j, \quad \text{with } \mathbb{E}[\epsilon_j] = 0 \tag{4}$$

where $\hat{a}_j$ is the normalized historical cumulative attention score of token j, c is a scaling constant, and ϵ_j is a zero-mean random noise term.

3.2 Different Treatment of Keys and Values

We first examine the fundamental difference between keys and values in the attention mechanism.

Keys vs. Values: Different Computational Spaces. Keys participates in dot products with the Query ($q \cdot k_j$) inside the Softmax exponent, where any perturbation is exponentially transformed into a multiplicative perturbation on attention scores. Values participates in the post-Softmax weighted sum $\sum a_j v_j$, a linear operation where errors propagate only linearly. This difference is a structural property of the Transformer architecture, independent of any particular merging algorithm. We now formalize this analysis.

Additive Keys Merging Produces Exponential Multiplicative Perturbations. Consider merging M keys vectors via linear combination $k_m = \sum_{j=1}^{M} \alpha_j k_j$. Let q_{real} and q_{est} denote the true and estimated future Queries, with deviation $\Delta q = q_{\text{real}} - q_{\text{est}}$ and per-key perturbation $\delta_j = \Delta q \cdot k_j$. The unnormalized attention score under the true Query is:

$$\begin{aligned} s_m &= \exp\left(q_{\text{real}} \cdot k_m\right) \\ &= \exp\left((q_{\text{est}} + \Delta q) \cdot \sum_{j=1}^{M} \alpha_j k_j\right) \end{aligned} \tag{5}$$

Expanding the dot product to separate the estimated term from the perturbation term yields:

$$\begin{aligned} s_m &= \exp\left(\sum_{j=1}^{M} \alpha_j (q_{\text{est}} \cdot k_j)\right) \cdot \exp\left(\sum_{j=1}^{M} \alpha_j \delta_j\right) \\ &= \hat{S}_{\text{target}} \cdot \underbrace{\exp\left(\sum_{j=1}^{M} \alpha_j \delta_j\right)}_{E_{\text{key}}} \end{aligned} \tag{6}$$

The first term $\hat{S}_{\text{target}}$ is the intended score under the attention consistency assumption. The second term $E_{\text{key}} = \exp(\sum \alpha_j \delta_j)$ is the error amplification factor. Any nonzero $\sum \alpha_j \delta_j$ is exponentially transformed into a multiplicative perturbation of attention scores. This holds for any additive keys merging scheme, including KeepKV's logarithmic correction. See Appendix A for a concrete case study. In summary, additive keys merging attempts to approximate an exponential-space mapping with linear structure. The approximation holds only in the degenerate case $q_{\text{real}} = q_{\text{est}}$, and any attention deviation causes exponential error amplification.

Robustness of Additive Values Merging. In contrast, values merging takes place after the Softmax. Let the merged values be $v_m = \sum_{j=1}^{M} \gamma_j v_j$, with $\sum_{j=1}^{M} \gamma_j = 1$. Given an attention score a_m, even if this score carries an error ϵ (i.e., $a_m = a_{\text{ideal}} + \epsilon$), the contribution of this token to the attention output is:

$$o_m = a_m \cdot v_m = a_{\text{ideal}} \cdot v_m + \epsilon \cdot \sum_{j=1}^{M} \gamma_j v_j \tag{7}$$

The error term is linear in ϵ with no nonlinear amplification, and bounded by:

$$\left\| \epsilon \cdot \sum \gamma_j v_j \right\| \leq |\epsilon| \cdot \max_j \|v_j\| \tag{8}$$

3.3 Expected Contribution Matching (ECM)

Having established the differentiated strategy of keys eviction + values merging, we now address how to perform the values merging. This involves two specific sub-problems: selecting the target set and designing weight allocation.

Problem Formalization. Consider a single compression operation. Let token i be selected for eviction. Its values v_i will be distributed across a target set T ($|T| = W$) of retained tokens according to weights $\mathbf{w} = [w_{i \to t}]_{t \in T}$. For any future Query, the reconstructed attention after compression is $\tilde{a}_i = \sum_{t \in T} w_{i \to t} a_t$. The deviation between this and the true attention a_i directly determines the magnitude of the compression error. Our goal is to find the optimal weights $\mathbf{w}$ that minimize the expected future reconstruction error. To encourage smooth weight allocation and avoid excessive concentration on a few tokens, we formulate the loss in Ridge regression form (Hoerl and Kennard, 2000):

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbb{E}\left[\left(\sum_{t \in T} w_{i \to t} a_t - a_i\right)^2\right] + \lambda \|\mathbf{w}\|^2 \tag{9}$$

where $\lambda > 0$ is the regularization coefficient. Substituting the attention consistency assumption $a_j \approx c \cdot \hat{a}_j$ (the constant c can be absorbed into the objective without changing the optimal solution), the problem simplifies to:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \left(\mathbf{w}^T \hat{\mathbf{a}}_T - \hat{a}_i\right)^2 + \lambda \|\mathbf{w}\|^2 \quad (10)$$

where $\hat{\mathbf{a}}_T = [\hat{a}_{t_1}, \ldots, \hat{a}_{t_W}]^T \in \mathbb{R}^W$ is the vector of normalized historical cumulative attention scores for the tokens in the target set.

ECM Closed-Form Solution. $\mathcal{L}(\mathbf{w})$ is a strictly convex quadratic function in $\mathbf{w}$. Its global minimum satisfies the zero-gradient condition:

$$\nabla_{\mathbf{w}} \mathcal{L} = 2(\mathbf{w}^T \hat{\mathbf{a}}_T - \hat{a}_i)\hat{\mathbf{a}}_T + 2\lambda \mathbf{w} = 0 \quad (11)$$

Rearranging gives $(\hat{\mathbf{a}}_T \hat{\mathbf{a}}_T^T + \lambda I)\mathbf{w} = \hat{a}_i \hat{\mathbf{a}}_T$. Note that $\hat{\mathbf{a}}_T \hat{\mathbf{a}}_T^T$ is a rank-1 matrix whose only nonzero eigenvector is $\hat{\mathbf{a}}_T$ itself. The optimal solution $\mathbf{w}^*$ must therefore be collinear with $\hat{\mathbf{a}}_T$. Setting $\mathbf{w}^* = \alpha \hat{\mathbf{a}}_T$ and solving yields $\alpha = \hat{a}_i / (\|\hat{\mathbf{a}}_T\|^2 + \lambda)$. This gives the closed-form optimal weights:

$$w^*_{i \to t} = \frac{\hat{a}_i \hat{a}_t}{\sum_{s \in T} \hat{a}_s^2 + \lambda} \quad (12)$$

This formula is the core of ECM. The weight $w^*_{i \to t}$ is proportional to the target's historical attention $\hat{a}_t$: tokens more likely to be hit by future Queries carry more of the evicted values' information, which is exactly the origin of the term "expected contribution matching". As the global optimum of a strictly convex objective (Boyd and Vandenberghe, 2014), ECM is provably superior to any alternative allocation (e.g., uniform $w_{i \to t} = 1/W$), with equality only in the degenerate case where all $\hat{a}_t$ are identical, which almost never occurs under the sparse attention distributions of LLMs. We provide experimental support for the advantage of the ECM weight allocation scheme in Section 4.

Unlike CaM (Zhang et al., 2024), which also merges only values, ECM differs in two key aspects: ECM uses attention-proportional weights rather than uniform averaging, and ECM is fully deterministic rather than relying on Bernoulli sampling.

Target Set Selection via Global Semantic Matching. In early works, CaM employs a local consecutive window strategy, merging the evicted values into the W consecutive tokens immediately following it. This approach is simple but completely ignores semantic relationships among tokens. Intuitively, merging values v_i into semantically unrelated targets causes semantic contamination. We adopt a global semantic matching strategy: computing the cosine similarity between the evicted value v_i and every retained value in the cache, and select the Top-W most similar ones as the target set T. The intuition is that merging information about a certain entity into semantically related tokens is less lossy than merging it into entirely unrelated ones. We provide experimental support for this strategy in Section 4.

Algorithm 1: KV Cache Compression with ECM

Input: KV Cache (K, V) of length L, cache budget B, sink budget S, merge width W, regularization λ

Output: Compressed KV Cache of length $B + S$

Select $L - B - S$ tokens as evict_set
for *each evicted token* $i \in$ *evict_set* **do**
  $T \leftarrow$ top-W indices by similarity$(V[i], V[j])$ for $j \notin$ evict_set
  for *each* $t \in T$ **do**
    $V[t] \leftarrow V[t] + \frac{\hat{a}[i] \cdot \hat{a}[t]}{\sum_{s \in T} \hat{a}[s]^2 + \lambda} \cdot V[i]$
  end
end
Remove $K[i]$ and $V[i]$ for all $i \in$ evict_set.
return compressed KV Cache;

3.4 Algorithm Summary

The complete asymmetric KV Cache compression pipeline is presented in Algorithm 1, with an illustrative overview shown in Figure 2. The algorithm consists of three stages: eviction selection (Phase 1), ECM values merging (Phase 2), and keys eviction and finalization (Phase 3).

Phase 1 (Eviction Selection) preserves the first S sink tokens, then scores tokens in the middle region by cumulative attention and selects the K_{evict} lowest-scoring tokens for eviction. **Phase 2 (ECM Merging)** selects the Top-W semantically closest retained tokens via values' cosine similarity for each evicted token, then computes ECM weights (Eq. 12) to redistribute the evicted value. **Phase 3 (Eviction)** removes the KV pairs of evicted tokens and trims cumulative attention records.

4 Experiments

In this section, we first introduce the experimental setup (Section 4.1), then present the main evaluation results on LongBench and LongBenchV2 (Section 4.2), followed by an analysis of performance trends under varying compression ratios in (Section 4.3). We then verify each of the claims proposed in methodology chapter through controlled experiments (Section 4.4), and finally analyze compression efficiency (Section 4.5).

4.1 Experimental Setup

Models. We evaluate on two mainstream open-source instruction-tuned LLMs: Llama-3.1-8B-Instruct (Team, 2024) (which uses GQA (Ainslie et al., 2023)) and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). Both models are widely adopted base models for long-context evaluation. (We use publicly available datasets and follow their original licenses and anonymization protocols. No personally identifying information was intentionally collected.)

Benchmarks. We use two long-text evaluation benchmarks. **LongBench** (Bai et al., 2024) comprises 14 English subtasks and two code tasks with input lengths ranging from 5K to 15K tokens, including Single-Document QA, Multi-Document QA, Summarization, Few-Shot Learning, Synthetic, and Code. **LongBenchV2** (Bai et al., 2025) targets even longer contexts (from 8K to 2M), emphasizing reasoning ability and evidence localization. It contains 503 high-quality annotated samples partitioned into subsets by difficulty and text length.

Baselines. We compare against the following Training-Free KV Cache compression methods. Eviction-based: **StreamingLLM** (retains sink tokens + recent window), **H2O** (retains Heavy Hitters based on cumulative attention). Merging-based: **CaM** (merges into a local consecutive window, Bernoulli sampling), **AsymKV** (Keys curvature-weighted merging + Values merging). In addition, **Full Cache** serves as the uncompressed upper bound.

Cache Configuration and Inference Details. The cache budget is uniformly set to 2048 plus the first 32 sink tokens. We use chunked prefill (Agrawal et al., 2023) (input_window = 512) during the Prefill stage to simulate realistic long-text inference. All experiments run on HuggingFace Transformers (Wolf et al., 2020) in eager attention mode. Evaluation metrics are the raw scores of each subtask. We report both category averages and the overall average. Compression time and memory footprint measurements are conducted on a single A6000-48GB GPU.

Method	S-Doc	M-Doc	Sum.	F-S	Syn.	Code	Avg.
Full Cache	45.19	46.30	28.81	69.81	53.72	57.97	50.29
SLLM	27.60	28.77	24.55	63.77	15.50	**60.54**	36.79
H2O	37.82	38.24	26.58	**66.10**	14.50	56.24	39.91
CaM	37.59	36.64	24.80	**66.10**	16.13	49.56	38.47
AsymKV	**39.51**	40.41	26.77	65.46	38.21	54.77	44.19
ECM	39.35	**43.90**	**27.01**	64.65	**44.00**	56.19	**45.85**

Table 1: Results on LongBench for Llama-3.1-8B-Instruct. The cache budget is 2048 plus 32 sink tokens. All methods are reproduced under the same experimental settings. **Bold** indicates the best result among compression methods, while Full Cache serves as the uncompressed upper bound. SLLM denotes StreamingLLM.

Method	S-Doc	M-Doc	Sum.	F-S	Syn.	Code	Avg.
Full Cache	38.63	37.79	29.00	70.45	50.75	60.12	47.79
SLLM	24.23	22.38	24.94	65.91	13.71	58.51	34.95
H2O	31.64	30.81	27.13	**68.66**	13.28	58.31	38.62
CaM	30.59	30.83	24.92	67.63	32.00	57.93	40.65
AsymKV	33.22	31.56	26.81	67.01	33.78	**58.55**	41.82
ECM	**33.42**	**33.54**	**27.36**	67.10	**34.31**	57.15	**42.14**

Table 2: Results on LongBench for Mistral-7B-Instruct-v0.3. Cache budget is 2048 physical KV pairs plus 32 sink tokens. **Bold** indicates the best result. ECM achieves the best performance on overall average. SLLM refers to StreamingLLM.

4.2 Main Results

LongBench Evaluation. Tables 1 and 2 report the results on LongBench across six categories for Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3, respectively.

On LongBench, ECM achieves the best average score on both models. For Llama-3.1-8B-Instruct, ECM scores 45.85, outperforming AsymKV by 1.66 points and the reproduced CaM baseline by 7.38 points. For Mistral-7B-Instruct-v0.3, ECM scores 42.14, versus 41.82 for AsymKV. Particularly strong gains are observed on Multi-Doc QA, where ECM outperforms AsymKV by 3.49 points on Llama, and on Synthetic tasks, where it leads by 5.80 points. Moreover, the merging-based methods ECM and AsymKV achieve higher overall averages than both eviction-based baselines on Llama-3.1-8B-Instruct.

LongBenchV2 Evaluation. Table 3 compares

Method	Overall	Easy	Hard	Short	Medium	Long
Full Cache	30.0	30.7	29.6	35.0	27.9	25.9
SLLM	25.8	25.0	26.4	35.0	19.1	**24.1**
H2O	26.6	26.0	27.0	**35.6**	24.2	16.7
CaM	27.0	26.6	27.3	**35.6**	24.7	17.6
AsymKV	28.2	**31.2**	26.4	31.1	**27.9**	**24.1**
ECM	**28.8**	28.6	**28.9**	34.4	26.5	**24.1**

Table 3: Results on LongBenchV2 for Llama-3.1-8B-Instruct. Cache budget is 2048 plus 32 sink tokens. **Bold** indicates the best result. Overall = average over all samples. Easy/Hard = partition by difficulty. Short/Medium/Long = partition by text length.

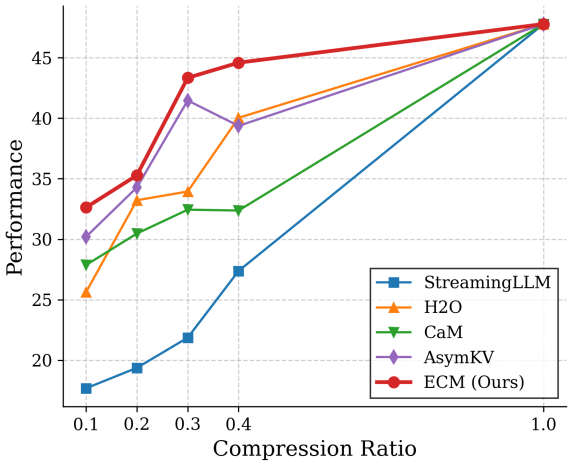


Figure 3: Performance comparison on 2wikimqa under different cache retention ratios. ECM consistently outperforms all other methods at every retention ratio.

ECM with both eviction-based and merging-based training-free baselines on LongBenchV2.

ECM achieves the highest overall score among the evaluated compression methods, reaching 28.8 versus 28.2 for AsymKV. Its largest advantage over AsymKV appears on the Hard subset, where it improves from 26.4 to 28.9. ECM is not uniformly best across all subsets: AsymKV leads on Easy and Medium, H2O and CaM tie on Short, and ECM ties with SLLM and AsymKV on Long.

4.3 Compression Ratio Sensitivity Analysis

To examine ECM's robustness under varying compression intensities, we conduct comparative experiments on 2wikimqa (a representative Multi-Doc QA subtask) under different cache budgets, with compression ratios ranging from 0.1 to 0.4, as shown in Figure 3.

ECM outperforms all compared methods at every compression ratio, with the largest margin around ratio 0.4 (44.58, exceeding others by >4 points and approaching Full Cache at 47.77). The advantage over AsymKV grows from 2 points at 30% cache budget to 5 points at 40%, reflecting the combined benefit of direct keys eviction, ECM weight allocation, and global semantic matching under higher compression.

Strategy	S-Doc	M-Doc	Sum.	F-S	Syn.	Code	Avg.
Full Cache	45.19	46.30	28.81	69.81	53.72	57.97	50.29
mK_mV	39.51	40.41	26.77	**65.46**	38.21	54.77	44.19
eK_mV	**39.61**	**42.37**	**26.78**	64.59	**41.92**	**55.23**	**45.08**

Table 4: Comparison between keys eviction and keys merging. Model: Llama-3.1-8B-Instruct. Benchmark: LongBench. Cache budget: 2048 + 32 sink tokens. mK_mV uses AsymKV's original keys gradient-weighted averaging + values merging. eK_mV uses direct keys eviction + values merging. The two settings differ only in keys' treatment. The values merging strategy is identical. **Bold** marks the best result in each column. eK_mV outperforms mK_mV on all tasks except Few-Shot.

4.4 Controlled Experiments

In this section, we verify each of the core design choices from the Methodology through three controlled experiments, namely whether keys eviction outperforms keys merging (Section 3.2), whether ECM merging outperforms uniform averaging (Section 3.3), and whether global semantic matching outperforms local consecutive windows (Section 3.3).

Keys Eviction vs. Keys Merging. Table 4 compares two keys treatments under identical values merging conditions: mK_mV (AsymKV's keys gradient-weighted averaging) and eK_mV (direct keys eviction). The values merging strategy is identical in both settings, using AsymKV's original direct summation with Softmax denominator correction, so that the only difference between the two settings is the treatment of keys.

Keys eviction outperforms keys merging on all tasks except Few-Shot, with notable gains on Multi-Doc QA (+1.96) and Synthetic (+3.71), supporting the theoretical analysis in Section 3.2. Moreover, eK_mV achieves this with higher efficiency, as direct keys eviction eliminates the need to compute information for merging, avoiding extra computational overhead.

ECM Weight Allocation vs. Uniform Averaging. To isolate weight allocation, we fix the target set to a local consecutive window and compare ECM vs. Uniform ($w_{i \to t} = 1/W$) on three

Strategy	2wikimqa	hotpotqa	musique	Avg.
Full Cache	47.77	58.03	33.09	46.30
Avg-local	43.54	54.01	27.30	41.62
ECM-local	45.18	53.11	27.86	42.05
ECM	**47.81**	**54.50**	**29.37**	**43.90**

Table 5: Controlled Comparisons of ECM Weight Allocation and Global Semantic Matching. Model: Llama-3.1-8B-Instruct. Evaluation data: 2wikimqa / hotpotqa / musique (Multi-Doc QA subtasks). **Bold** = best, underline = second best. The three rows correspond to: (1) Avg-local: local consecutive window + uniform weight allocation ($w_{i\to t} = 1/W$). (2) ECM-local: local consecutive window + ECM attention-proportional weight allocation. (3) ECM: global semantic matching + ECM weight allocation, i.e., our complete method.

Multi-Doc QA subtasks. These correspond to rows "Avg-local" and "ECM-local" in Table 5.

ECM weight allocation outperforms uniform averaging across three datasets, with the largest gain on 2wikimqa (+1.64).

Global Semantic Matching vs. Local Windows. We compare global semantic matching against local consecutive windows, with both using ECM weight allocation (rows "ECM-local" vs. "ECM" in Table 5). Global matching exceeds local windows by 1.85 points on average. On 2wikimqa, ECM with global matching (47.81) even slightly surpasses Full Cache (47.77). A plausible explanation is that global semantic matching acts as feature enhancement, by transferring information to the semantically most relevant values.

4.5 Efficiency Analysis

We evaluate compression latency and GPU memory footprint on a single A6000-48GB GPU with Llama-3.1-8B-Instruct (cache budget: 2048 + 32 sink tokens).

Compression Latency. Figure 4 shows compression overhead at the Prefill stage. ECM's main overhead comes from cosine similarity computation during global semantic matching, with complexity $O(K \cdot L \cdot d)$, where K is the number of evicted tokens, L is the sequence length, and d is the values vector dimension. In contrast, AsymKV is approximately 40% slower than ECM.

GPU Memory Footprint. Table 6 reports peak memory. ECM (23,070 MB) is substantially lower than AsymKV (33,710 MB), as ECM uses only forward computation while AsymKV retains intermediate activations for backpropagation.

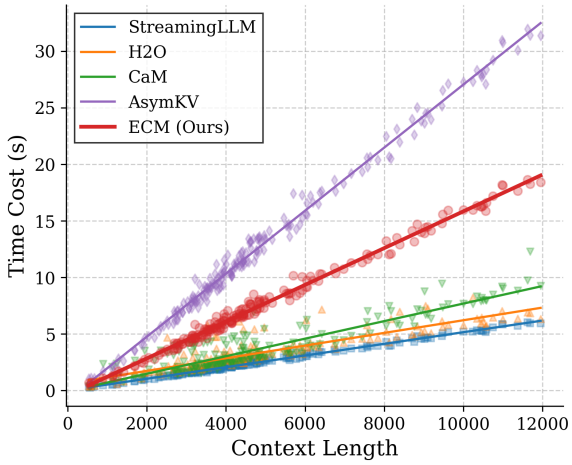


Figure 4: Compression latency at Prefill. ECM's overhead comes from global semantic matching. AsymKV is slower due to Hessian matrix approximation.

	SLLM	H2O	CaM	AsymKV	ECM
Memory	16,954	16,581	21,988	33,710	23,070

Table 6: Peak GPU memory footprint (MB). Model: Llama-3.1-8B-Instruct, A6000-48GB, dataset: 2wikimqa (chunked prefill, input_window=512).

5 Conclusion

We revealed a fundamental asymmetry between keys and values in KV Cache merging: keys vectors operate inside the Softmax exponential space, making additive merging structurally unreliable, while values vectors in the post-Softmax linear space admit bounded errors. Based on this insight, we proposed a differentiated strategy of keys eviction + values merging, derived ECM as a provably optimal weight allocation for values merging, and paired it with global semantic matching for target selection. On LongBench and LongBenchV2, ECM achieves the highest overall averages among the evaluated training-free methods under the same cache budget, and controlled experiments confirm the contribution of each design choice. Supplementary evaluations on Qwen2.5-14B-Instruct and RULER 4K are reported in Appendix B.

Limitations

Attention consistency assumption. Similar to typical merging-based methods, ECM relies on historical attention to predict future attention. A context distribution shift weakens this correlation.

Global semantic matching complexity. Cosine similarity search incurs $O(K \cdot L \cdot d)$ cost,

which may become a bottleneck for extremely long contexts. Future work could explore approximate nearest-neighbor search or hierarchical clustering to accelerate the matching process.

Model scale. Our experiments are limited to 7B/8B/14B models due to computational constraints. Whether the conclusions hold for larger models remains to be verified.

Acknowledgments

We gratefully acknowledge the support of the National Natural Science Foundation of China (NSFC) via grant 62236004 and 62476073.

References

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. 2023. SARATHI: efficient LLM inference by piggybacking decodes with chunked prefills. *CoRR*, abs/2308.16369.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. 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. Association for Computational Linguistics.

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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137. Association for Computational Linguistics.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 3639–3664. Association for Computational Linguistics.

Stephen P. Boyd and Lieven Vandenberghe. 2014. *Convex Optimization*. Cambridge University Press.

Ngoc Bui, Shubham Sharma, Simran Lamba, Saumitra Mishra, and Rex Ying. 2025. Cache what lasts: Token retention for memory-bounded KV cache in llms. *CoRR*, abs/2512.03324.

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

Wanyun Cui and Mingwei Xu. 2025. Homogeneous keys, heterogeneous values: Exploiting local KV cache asymmetry for long-context llms. *CoRR*, abs/2506.05410.

Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. 2025. Rethinking key-value cache compression techniques for large language model serving. In *Proceedings of the Eighth Conference on Machine Learning and Systems, MLSys 2025, Santa Clara, CA, USA, May 12-15, 2025*. OpenReview.net/mlsys.org.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Arthur E. Hoerl and Robert W. Kennard. 2000. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. 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. ACM.

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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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*.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. 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*, Proceedings of Machine Learning Research, pages 47901–47911. PMLR / OpenReview.net.

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

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. 2026. Keepkv: Achieving periodic lossless KV cache compression for efficient LLM inference. In *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 33259–33267. AAAI Press.

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*, pages 5998–6008.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. 2025. 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*. OpenReview.net.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 58840–58850. PMLR / OpenReview.net.

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 Perturbation Analysis of Additive Keys Merging under Attention Deviation

This appendix uses the keys merging scheme from KeepKV (Tian et al., 2026) as a concrete case study. We show how an additive keys merging method that claims to be "lossless" under the attention consistency assumption produces exponential multiplicative perturbations when attention deviation is present. The core argument: **additive keys merging cannot recover the true uncompressed attention structure under attention deviation**. This is a structural defect of any additive keys merging scheme, not a numerical precision issue. The analysis in this appendix provides a concrete instance supporting the general claim in Section 3.2 that "any additive keys merging produces exponential multiplicative perturbations under attention deviation."

A.1 KeepKV's Keys Merging Scheme

KeepKV proposes ZIP-Merging, deriving a keys merging formula claimed to be lossless under the attention consistency assumption (i.e., the EMA-estimated attention scores $\hat{s}_i$ equal the true future attention scores s_i).

Let the two tokens to be merged be e (Evict) and c (Retain). Under the attention consistency assumption (where the estimated Query q_{ema} equals the true Query), the estimated unnormalized attention scores are $\hat{s}_i = \exp(q_{\text{ema}} \cdot k_i/\sqrt{d})$, with corresponding logits $\hat{x}_i = q_{\text{ema}} \cdot k_i/\sqrt{d}$.

KeepKV merges values using standard weighted averaging:

$$v_r = \frac{\hat{s}_e v_e + \hat{s}_c v_c}{\hat{s}_e + \hat{s}_c} \tag{13}$$

For keys merging, it introduces a logarithmic scaling term to achieve losslessness under the attention consistency assumption. Simplifying by

setting the voting weights $p_e = p_c = 1$ (the full form appears in KeepKV Eq. 24–25):

$$k_r = \frac{\hat{s}_e k_e + \hat{s}_c k_c}{\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c} \ln(\hat{s}_e + \hat{s}_c) \tag{14}$$

When $q_{\text{real}} = q_{\text{ema}}$, one can verify that the merged k_r produces an attention score in the Softmax computation exactly equal to the sum of the attention scores of the two pre-merge tokens, thus achieving "losslessness."

A.2 Perturbation Propagation under Attention Deviation: Structural Exponential Error

Now consider the realistic inference scenario: a deviation $\Delta q = q_{\text{real}} - q_{\text{ema}}$ exists between the true future Query q_{real} and the estimated Query q_{ema} used to compute the merging weights. Define the dot-product perturbation on each token as $\Delta_e = \Delta q \cdot k_e$, $\Delta_c = \Delta q \cdot k_c$. The true logits are then $x_i = \hat{x}_i + \Delta_i$.

Taking the dot product of the true Query q_{real} with the merged k_r yields the true logit x_r:

$$x_r = q_{\text{real}} \cdot k_r \tag{15}$$

$$= \frac{\hat{s}_e(q_{\text{real}} \cdot k_e) + \hat{s}_c(q_{\text{real}} \cdot k_c)}{\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c} \ln(\hat{s}_e + \hat{s}_c) \tag{16}$$

$$= \frac{\hat{s}_e(\hat{x}_e + \Delta_e) + \hat{s}_c(\hat{x}_c + \Delta_c)}{\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c} \ln(\hat{s}_e + \hat{s}_c) \tag{17}$$

$$= \ln(\hat{s}_e + \hat{s}_c) + \underbrace{\frac{\ln(\hat{s}_e + \hat{s}_c)}{\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c}}_{C} \cdot (\hat{s}_e \Delta_e + \hat{s}_c \Delta_c) \tag{18}$$

The merged unnormalized attention score $s_r = \exp(x_r)$ is then:

$$s_r = (\hat{s}_e + \hat{s}_c) \cdot \exp\left(C \cdot (\hat{s}_e \Delta_e + \hat{s}_c \Delta_c)\right) \tag{19}$$

The key comparison is between s_r and the sum of the true attention scores of the two uncompressed tokens:

$$s_e + s_c = \hat{s}_e \exp(\Delta_e) + \hat{s}_c \exp(\Delta_c) \tag{20}$$

These two quantities are structurally different. s_r is a **single exponential term**: it places the weighted combination $\hat{s}_e \Delta_e + \hat{s}_c \Delta_c$ of Δ_e and Δ_c inside a single $\exp(\cdot)$. In contrast, $s_e + s_c$ is the **sum of two independent exponential terms**: Δ_e and Δ_c each independently enter the exponential function before being summed. These two forms are equal only when $\Delta_e = \Delta_c = 0$ (i.e., when the attention consistency assumption holds perfectly). Whenever $\Delta_e \neq \Delta_c$, a single exponential term $\exp(\hat{s}_e \Delta_e + \hat{s}_c \Delta_c)$ **cannot** exactly recover the sum of two independent exponential terms $\hat{s}_e \exp(\Delta_e) + \hat{s}_c \exp(\Delta_c)$.

This structural difference is not specific to KeepKV. It is an inherent limitation of any additive keys merging scheme: the merging operation compresses multiple keys vectors into a single keys vector, which means compressing multiple independent attention scores into a single attention score. No matter how the logarithmic scaling term or other correction terms are designed, the independent perturbation Δ_e of a merged-away token cannot be independently expressed within a single exponential term. This is the fundamental difference between keys merging and values merging. The output after values merging is $s_c \cdot v_r = s_c \cdot \frac{\hat{s}_e v_e + \hat{s}_c v_c}{\hat{s}_e + \hat{s}_c}$, where both values vectors v_e and v_c remain as independent additive terms and do not lose their independence through merging.

A.3 Numerical Instability: Error Amplification in Extreme Cases

The structural error described above is governed in general by the magnitudes of the attention deviations Δ_e, Δ_c. However, the perturbation amplifier $C = \frac{\ln(\hat{s}_e + \hat{s}_c)}{\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c}$ can further amplify this error in extreme cases.

The denominator $\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c$ is a scalar determined by the estimated Query q_{ema}. Since the logit $\hat{x}_i = q_{\text{ema}} \cdot k_i$ is the dot product of a keys vector and a Query in d-dimensional space, its range is $(-\infty, +\infty)$. When $\hat{x}_e$ and $\hat{x}_c$ have opposite signs with $\hat{s}_e|\hat{x}_e| \approx \hat{s}_c|\hat{x}_c|$, or when both $\hat{x}_e$ and $\hat{x}_c$ are close to zero (both keys are approximately orthogonal to the estimated Query), the denominator approaches zero and $C \to \infty$. The structural error described in Section A.2 already exists. In these extreme cases, the divergence of C further amplifies tiny Δ_e, Δ_c exponentially, completely distorting the attention score at that position. Notably, these extreme cases are not merely hypothetical: in the high-dimensional attention space of LLMs, when the Query falls in certain directions within the plane spanned by two keys vectors, $\hat{s}_e \hat{x}_e + \hat{s}_c \hat{x}_c \approx 0$ occurs with non-negligible probability.

In summary, additive keys merging faces a double risk: (1) under normal conditions, a structural

Method	2WikiMQA	HotpotQA	MuSiQue	NrtvQA	MFQA-en	Qasper	TREC	TriviaQA	SAMSum	Avg.
Full Cache	59.39	64.60	38.29	31.11	53.54	46.20	78.00	90.20	51.08	56.93
AsymKV	53.14	**56.16**	26.98	21.35	47.08	37.28	**71.50**	**88.61**	46.77	49.87
ECM	**53.29**	56.00	**33.78**	**28.07**	**47.36**	**39.36**	71.00	88.57	**47.31**	**51.64**

Table 7: Results on nine LongBench subtasks using Qwen2.5-14B-Instruct with a cache budget of 2048 plus 32 sink tokens. The evaluated subset contains three multi-document QA tasks, three single-document QA tasks, two few-shot learning tasks, and one summarization task. **Bold** indicates the best result among compression methods; Full Cache is the uncompressed upper bound.

Method	s1	s2	s3	mk1	mk2	mk3	mv	mq	vt	cwe	fwe	qa1	qa2	Avg.
Full Cache	100.00	100.00	100.00	100.00	100.00	99.80	99.70	99.65	99.96	98.96	86.40	84.80	61.80	94.70
AsymKV	**100.00**	99.20	**22.00**	**97.20**	68.60	38.80	92.80	**97.10**	**100.00**	**97.28**	**87.47**	**83.40**	**59.20**	80.23
ECM	**100.00**	**99.80**	19.40	97.00	**71.60**	**40.60**	**95.60**	96.90	**100.00**	96.86	86.40	**83.40**	59.00	**80.50**

Table 8: Results on RULER 4K using Llama-3.1-8B-Instruct with a cache budget of 2048 plus 32 sink tokens. **Bold** indicates the best result among compression methods; Full Cache is the uncompressed upper bound.

approximation error exists between a single exponential term and the sum of two independent exponential terms (Section A.2). (2) in extreme cases, the divergence of the perturbation amplifier C exponentially magnifies this error (Section A.3). Together, these demonstrate that additive keys merging is a highly unreliable operation under attention deviation.

A.4 Contrast: Error Propagation in Keys Eviction + Values Merging

Under the keys eviction + values merging strategy, keys of the evicted token e is directly deleted, retaining only k_c. The attention score at this position becomes $s_c = \hat{s}_c \exp(\Delta_c)$. Combined with the merged value v_r, its contribution to the attention output is:

$$o_{\text{evict+merge}} = s_c \cdot v_r = \hat{s}_c \exp(\Delta_c) \cdot \frac{\hat{s}_e v_e + \hat{s}_c v_c}{\hat{s}_e + \hat{s}_c} \tag{21}$$

This expression contains no perturbation amplifier C. The output error is scaled by $\exp(\Delta_c)$, and its norm is bounded by the norms of v_e and v_c.

A.5 Summary

By introducing a logarithmic scaling term, one can achieve lossless keys merging under the attention consistency assumption. However, this losslessness does not hold in realistic inference (when the attention consistency assumption is violated and a deviation exists between estimated and true attention). The analysis in this appendix reveals a **structural defect** of additive keys merging: the merging operation compresses multiple independent keys into a single key, replacing a sum of multiple independent exponential terms ($\hat{s}_e \exp(\Delta_e) + \hat{s}_c \exp(\Delta_c)$) with a single exponential term ($\exp(C \cdot (\hat{s}_e \Delta_e + \hat{s}_c \Delta_c))$). These two forms are structurally different whenever $\Delta_e \neq \Delta_c$, and this difference cannot be eliminated by logarithmic scaling or any other correction. It is the inevitable cost of the dimensionality reduction incurred by compressing multiple vectors into one. In extreme cases, the numerical divergence of the perturbation amplifier C further magnifies this structural error exponentially.

This appendix directly supports the core argument of Section 3.2: keys vectors lie inside the exponential space of Softmax, so additive merging attempts to approximate a sum of independent exponential terms with a single term—an approximation that inevitably deviates under attention shift. Hence, evicting keys and merging only values is the more robust strategy. In the values-merged output, each merged value remains an independent additive term, with errors propagating only linearly. The structural defect of keys merging does not arise.

B Supplementary Experimental Results

B.1 Evaluation on Qwen2.5-14B-Instruct

We evaluate Full Cache, AsymKV, and ECM on nine LongBench subtasks using Qwen2.5-14B-Instruct under the same cache configuration as the main experiments. Table 7 reports the results. ECM achieves an average score of 51.64, outperforming AsymKV by 1.77 points and retaining 90.7% of the Full-Cache score of 56.93. It outperforms AsymKV on six of the nine evaluated subtasks. Since this experiment covers only one 14B model and a subset of LongBench, it provides initial evidence of model-scale generalization rather than a complete larger-model evaluation.

B.2 Evaluation on RULER 4K

We additionally compare the three methods on the 13 RULER 4K subtasks using Llama-3.1-8B-Instruct. Table 8 reports the results under a cache configuration of 2048 plus 32 sink tokens. ECM obtains an average score of 80.50, compared with 80.23 for AsymKV, and retains 85.0% of the Full-Cache average of 94.70. The two compression methods are close on average and exhibit task-dependent differences. Both show substantial degradation on s3, mk2, and mk3 relative to Full Cache, indicating that the current compression setting remains challenging for some retrieval configurations.