

KeepKV: Achieving Periodic Lossless KV Cache Compression for Efficient LLM Inference

**Yuxuan Tian¹, Zihan Wang¹, Yebo Peng¹, Aomufei Yuan¹, Zhiming Wang¹,
Bairen Yi², Xin Liu², Yong Cui³, Tong Yang^{1*}**

¹State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University

²ByteDance

³Tsinghua University

Abstract

Efficient inference of large language models (LLMs) is hindered by an ever-growing key-value (KV) cache, making KV cache compression a critical research direction. Traditional methods selectively evict less important KV cache entries, which leads to information loss and hallucinations. Recently, merging-based strategies have been explored to retain more information by merging KV pairs that would be discarded; however, these existing approaches inevitably introduce inconsistencies in attention distributions before and after merging, causing degraded generation quality. To overcome this challenge, we propose KeepKV, a novel adaptive KV cache merging method designed to preserve performance under strict memory constraints, achieving single-step lossless compression and providing error bounds for multi-step compression. KeepKV introduces the Electoral Votes mechanism that records merging history and adaptively adjusts attention scores. Moreover, it further leverages a novel Zero Inference-Perturbation Merging method, compensating for attention loss resulting from cache merging. Extensive experiments on various benchmarks and LLM architectures demonstrate that KeepKV substantially reduces memory usage while successfully retaining essential context information, achieving over 2 \times inference throughput improvement and maintaining superior generation quality even with only 10% KV cache budgets.

1 Introduction

Transformer-based large language models (LLMs) have demonstrated remarkable capabilities across various applications (Touvron et al. 2023b; Jiang et al. 2023; OpenAI et al. 2024; Wan et al. 2024a; Rozière et al. 2024). To accelerate inference, LLMs commonly employ a key-value (KV) cache mechanism, which stores the KV embeddings of previously processed tokens to avoid redundant computations (Vaswani et al. 2017; Dai et al. 2019; Rae et al. 2019). However, as LLMs continue to support increasingly longer context lengths, the size of the KV cache grows rapidly, becoming a major bottleneck for inference (Kwon et al. 2023). For example, in the case of LLaMA-3-70B, a batch size of 128 with an 8K context length requires up to 320GB of KV cache memory (Grattafiori, Dubey et al. 2024). Consequently, com-

pressing the KV cache while preserving generation quality has become a crucial challenge.

Prior works mainly explore two approaches for KV cache compression: eviction-based and merging-based methods, both of which are inherently lossy. Eviction-based approaches selectively retain critical cache entries using heuristics like attention scores and token positions, permanently discarding less critical entries (Xiao et al. 2024b; Zhang et al. 2023; Reid and Zhu 2024; Liu et al. 2024; Li et al. 2024; Yang et al. 2024a) and thus causing context loss and potential hallucinations (Zhang et al. 2024). In contrast, merging-based methods aim to integrate rather than discard KV entries to retain more information. Recent representative studies, such as CaM (Zhang et al. 2024), D2O (Wan et al. 2024b), and KVMerger (Wang et al. 2024), have explored strategies like weighted key-value merging to mitigate context loss. Nevertheless, these methods vary widely in merge candidate selection and merging weight computation, and lack solid theoretical foundations. We observe that existing strategies inevitably induce attention inconsistencies and output perturbation. Specifically, the merged KV pair’s attention score is lower than the sum of the original scores prior to merging—a phenomenon we term “Attention Sag” (illustrated in Figure 1). These issues underline the necessity for an efficient and theoretically grounded KV cache merging strategy.

In this paper, we propose KeepKV, a novel KV cache merging method designed to maintain inference consistency and preserve essential contextual information. To the best of our knowledge, KeepKV is the first approach to achieve single-step lossless compression and to provide theoretical error bounds for multi-step compression. We first conduct a comprehensive theoretical analysis of existing eviction and weighted merging methods, grounded in the attention computation process, to reveal their fundamental limitations. Building on these theoretical insights, we propose a two-stage innovative design in KeepKV:

First, we propose techniques that achieve lossless compression for a single step. Specifically, we introduce the Electoral Votes mechanism, which records merging history, enabling accurate reconstruction of the original KV embeddings from compressed representations. Additionally, we present the Zero Inference-Perturbation Merging (ZIP-Merging) approach, which automatically adjusts weights to compensate for any losses caused by merging, maintaining attention con-

*Corresponding author.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

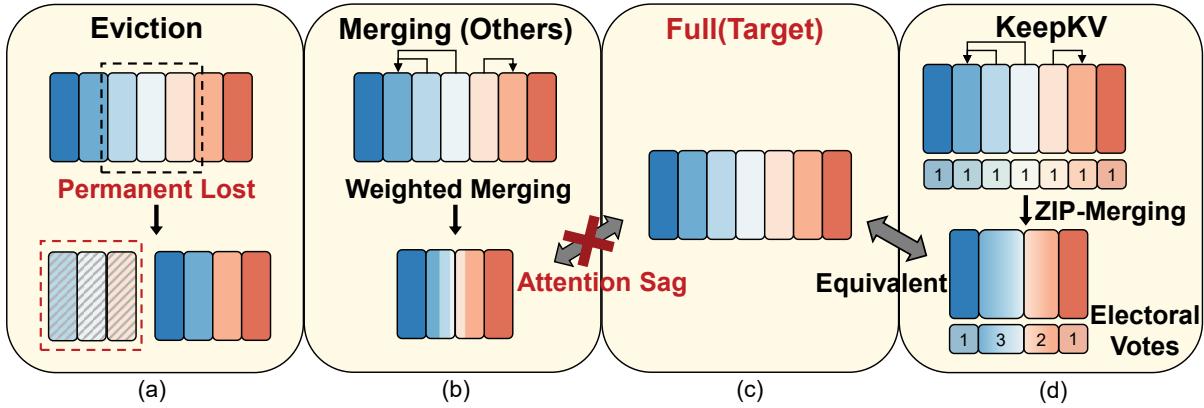


Figure 1: Illustration of KeepKV vs. Existing Methods. The three middle blocks represent KV subject to eviction/merging. (a) Eviction methods permanently discard them. (b) Merging methods integrates them into retained KV, but the result is not equivalent to the full KV, causing "Attention Sag." (c) Full KV serves as the ideal baseline. (d) KeepKV uses Electoral Votes as merging records and applies ZIP-Merging to minimize output disturbance, ensuring consistency and improving performance.

sistency. These designs theoretically guarantee zero output perturbation at the current iteration despite compression.

Second, we extend KeepKV to multi-step generation by estimating attention scores based on historical patterns. This is motivated by our empirical observation of strong locality in attention scores, also confirmed in prior studies (Dong et al. 2024; Zhang et al. 2024). Crucially, we provide theoretical analyses guaranteeing bounded output perturbation across multiple steps, thereby ensuring consistent inference quality under extended generation. Moreover, we offer a theoretical interpretation for prevalent similarity-based candidate selection methods, incorporating it into our design.

By integrating these innovations, KeepKV further enables periodic lossless KV cache compression by storing a complete KV cache externally and periodically loading compressed representations into memory, thereby maintaining inference consistency and preserving essential contextual information. Through theoretical derivation and extensive experiments, we demonstrate that KeepKV effectively preserves attention stability and output consistency, outperforming state-of-the-art KV cache eviction and merging methods. The contributions of this paper are summarized as follows:

- We propose KeepKV, a novel adaptive KV cache merging approach, which introduces the Electoral Votes mechanism and Zero Inference-Perturbation Merging, achieving single-step lossless compression and providing error bounds for multi-step compression.
- Extensive experiments across various tasks and models show that KeepKV maintains better performance under limited cache, outperforming existing KV cache eviction and merging methods.
- We are the first to theoretically analyze KV merging from the perspective of eliminating output perturbation. We provide guarantees on the perturbation bound of KeepKV and reveal the theoretical basis for merge candidate selection and weight design. Hopefully, our study can inspire future research on KV cache compression.

2 Related Work

KV cache has become a major bottleneck for efficient LLMs inference. Post-training optimization serves as a key solution due to its real-time and extensible capabilities.(Shi et al. 2024). Existing methods fall into three categories: **quantization**, **eviction**, and **merging**.

KV Cache Quantization. Quantization methods convert tensor values to lower precision to reduce bit-width. KVQuant (Hooper et al. 2024) applies Per-Channel Quantization for keys and Per-Token Quantization for values. MiKV (Yang et al. 2024b) introduces mixed-precision KV caching, where less critical KV are stored at lower precision. Additionally, GEAR (Kang et al. 2024) leverages low-rank matrix approximation for quantization residuals to minimize quantization loss. Our KeepKV reduces the number of cached KV pairs through merging, which is orthogonal to quantization methods and can be combined for better efficiency.

KV Cache Eviction. Eviction methods only retain more important KV entries. StreamingLLM (Xiao et al. 2024b) and LM-infinite (Han et al. 2024) identifies the importance of the initial k tokens for generation. H2O (Zhang et al. 2023), ScissorsHand (Liu et al. 2024) and RoCo (Reid and Zhu 2024) recognize crucial KV based on attention scores, while SnapKV (Li et al. 2024) utilizes attention within an observation window. Recent works explore improved budget allocation strategies across layers and heads. Pyramid (Cai et al. 2024; Yang et al. 2024a) allocates more cache to lower layers, whereas AdaKV (Feng et al. 2024), HeadKV (Fu et al. 2024), and DuoAttention (Xiao et al. 2024a) focus on inter-head differences. However, eviction causes irreversible information loss, potentially degrading generation quality.

KV Cache Merging. KV cache merging combines less important KV entries instead of permanently discarding them. DMC (Nawrot et al. 2024) learns when and how to merge through training, which limits generalization and introduces extra overhead. In contrast, CaM (Zhang et al. 2024) adaptively merges evicted value states into others but does not merge the corresponding keys. Recently, D2O (Wan

et al. 2024b) selects merge candidates and assigns merging weights based on cosine similarity between key states, while KVMerger (Wang et al. 2024) introduces a clustering-based method to group merge candidates and computes merging weights using Gaussian Kernel Weights. However, these methods fail to maintain attention consistency before and after merging, leading to output perturbation. We propose a novel merging approach designed to eliminate output perturbation, supported by theoretical analysis and extensive comparisons.

3 Methodology

3.1 Preliminary: Inference with KV Cache

We first introduce the attention computation process with KV cache. For simplicity, we consider a single attention head at one layer. Let the attention module’s weight matrices be $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$, where d denotes the hidden dimension. During the prefill stage, given an input prompt tensor $X_L \in \mathbb{R}^{L \times d} = [x_1, x_2, \dots, x_L]$, where L represents the prompt length, the KV states are computed and stored in the KV cache as follows:

$$\begin{aligned} K_L &= X_L W_k = [k_1, k_2, \dots, k_L], \\ V_L &= X_L W_v = [v_1, v_2, \dots, v_L]. \end{aligned} \quad (1)$$

In the decoding phase, KV cache are repeatedly utilized, while the newly computed KV pairs are continuously appended to it. Specifically, given the input at the t -th generation step, $x_t \in \mathbb{R}^d$, the KV cache update and attention computation are performed as follows:

$$\begin{aligned} K_t &= [K_{t-1}, k_t], \quad k_t = x_t W_k \\ V_t &= [V_{t-1}, v_t], \quad v_t = x_t W_v \end{aligned} \quad (2)$$

$$\begin{aligned} A^t &= \text{softmax} \left(\frac{q_t K_t^T}{\sqrt{d}} \right), \quad q_t = x_t W_q \\ s_i^t &= e^{\frac{q_t k_i}{\sqrt{d}}}, \quad o_t = \sum_{i=1}^t A_i^t v_i = \frac{\sum_{i=1}^t s_i^t v_i}{\sum_{i=1}^t s_i^t} \end{aligned} \quad (3)$$

KV cache effectively reduces redundant computation, but at the cost of increased memory consumption. Therefore, an important challenge is to compress the KV cache while maintaining model performance.

3.2 Rethinking KV Cache Eviction and Merging

Eviction and merging methods reduce memory usage by decreasing the number of stored KV pairs. The core motivation behind these studies is to minimize the impact of cache compression on the output. A fundamental *subtask* is to ensure that the output (o_t) remains as close as possible before and after compression at the current step. However, our analysis shows that existing methods inevitably introduce output perturbation and can not accomplish this task.

Perturbation in KV Cache Eviction. Eviction methods discard KV pairs deemed unimportant. Suppose we discard the pair (k_e, v_e) , and denote the output as o'_t . Based on Equation 3, we obtain:

$$o'_t = \frac{\sum_{i=1, i \neq e}^t s_i^t v_i}{\sum_{i=1, i \neq e}^t s_i^t} = \frac{1}{1 - A_e^t} (o_t - A_e^t v_e). \quad (4)$$

Remark 1. Equation 4 reveals that evicting (k_e, v_e) causes o'_t to deviate from o_t , with the deviation primarily determined by A_e^t . This formally explains why eviction methods generally prioritize discarding KV pairs with lower attention scores.

Although current methods optimize eviction and cache allocation strategies (Yang et al. 2024a; Fu et al. 2024) to minimize output impact, they cannot eliminate the perturbation in Equation 4. Previous studies have indicated that attention is not always sparse, especially in tasks requiring full context, as shown in Figure 2. Moreover, evicted KV may become important later, but irreversible eviction leads to permanent loss.

Attention Sag in KV Cache Merging. Merging methods integrate less important KV into others rather than discarding them. Existing studies typically use weighted merging (Nawrot et al. 2024; Wan et al. 2024b; Wang et al. 2024); formally, merging (k_e, v_e) into (k_c, v_c) is expressed as:

$$k_r = w_e k_e + w_c k_c, \quad v_r = w_e v_e + w_c v_c. \quad (5)$$

Here, (k_r, v_r) are the merged vectors, with weights w_e, w_c determined by the merging method. In D2O (Wan et al. 2024b), they depend on the cosine similarity between k_e and k_c , while in KVMerger (Wang et al. 2024), they are computed using Gaussian Kernel values. The weights satisfy the normalization condition $w_e + w_c = 1$. However, this widely used convex combination method also introduces output perturbations:

Theorem 2. Current weighted merging (convex combination) methods reduce the merged KV pair’s attention score compared to the sum of the original scores before merging, i.e., $A_r^t < A_e^t + A_c^t$, ultimately leading to $\|o'_r - o_t\| > 0$.

The formal proof is in Appendix A.2. We term this attention inconsistency from merging as **Attention Sag** and Figure 2 (c) illustrates this phenomenon. We provide an intuitive comprehension: existing methods merge multiple vectors into one, treating it equivalently as any other single vector in subsequent attention computations. This erases merging history, making it impossible to distinguish whether a KV pair is original or has absorbed numerous others.

3.3 Method: KeepKV

Electoral Votes and ZIP-Merging **Electoral Votes.** To address Attention Sag, we propose the Electoral Votes mechanism, which records the number of merges p_i (initialized to 1) each KV pair undergoes. A natural analogy is the Electoral College system, where electors hold votes proportional to their state’s population rather than a uniform share. The attention score of each KV is then scaled by its votes to approximate the original multiple KV’s influence before merging. For example, if a KV pair (k_r, v_r) has a vote count of $p_r = 3$, it is equivalent to three identical and independent instances of (k_r, v_r) participating in the attention computation. Formally,

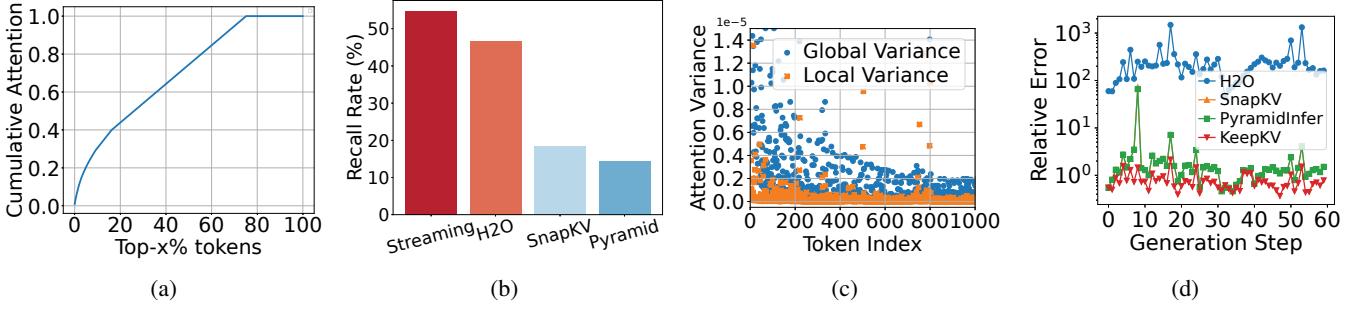


Figure 2: (a) Cumulative distribution of attention scores. Retaining the top- k tokens does not always preserve the majority of scores. (b) Proportion of to-be-evicted prompt tokens appearing in the top-20% attention scores during generation (compression rate = 20%). (c) Each token’s variance of its attention scores at each generation step (blue dots) is greater than the average variance within a sliding window (orange dots).(d) Relative errors for prediction of KeepKV and existing methods.

the outputs before (o_t) and after merging (o'_t) are defined as follows:

$$\begin{aligned} o_t &= \frac{\sum_{i=1}^t p_i s_i^t v_i}{\sum_{i=1}^t p_i s_i^t}, \\ o'_t &= \frac{\sum_{i=1, i \neq e, c}^t p_i s_i^t v_i + p_r s_r^t v_r}{\sum_{i=1, i \neq e, c}^t p_i s_i^t + p_r s_r^t}, \\ p_r &= p_e + p_c. \end{aligned} \quad (6)$$

Zero Inference-Perturbation Merging (ZIP-Merging). The Electoral Votes mechanism enables the elimination of output perturbations. We define the merging equations and theorem as follows:

$$\begin{aligned} k_r &= \frac{(w_e k_e + w_c k_c) \ln \frac{w_e + w_c}{p_e + p_c}}{w_e \ln s_e^t + w_c \ln s_c^t}, \\ v_r &= \frac{w_e v_e + w_c v_c}{w_e + w_c}, \\ w_e &= p_e s_e^t, \quad w_c = p_c s_c^t. \end{aligned} \quad (7)$$

Theorem 3. *The merging method in Equation 7 is perturbation-free, that is, $\|o'_t - o_t\| = 0$*

Remark 4. *The proof is in Appendix A.3. Intuitively, our method ensures attention consistency by preserving historical information via Electoral Votes and applying proper scaling (ZIP-Merging) to (k_r, v_r) instead of a convex combination.*

This theorem confirms that our novel merging approach can eliminate output perturbations and complete the subtask introduced at the beginning of this section. However, its applicability remains limited to the current iteration, and extending it to multi-step generation requires additional design.

Extending to Multi-Step Generation EMA Attention Scores. For ZIP-Merging to be effective in real-world multi-step generation, a solid comprehension of attention score dynamics is essential. Fortunately, empirical observations show that attention scores exhibit strong locality (Figure 2 (d)), meaning a token’s attention scores evolve smoothly across adjacent steps, which is also validated by prior studies

(Yang et al. 2024a; Zhang et al. 2024; Dong et al. 2024). From this, we employ the Exponential Moving Average (EMA) (Hunter 1986; Busbridge et al. 2023) with bias correction, a widely used technique in time-series analysis, formulated as follows:

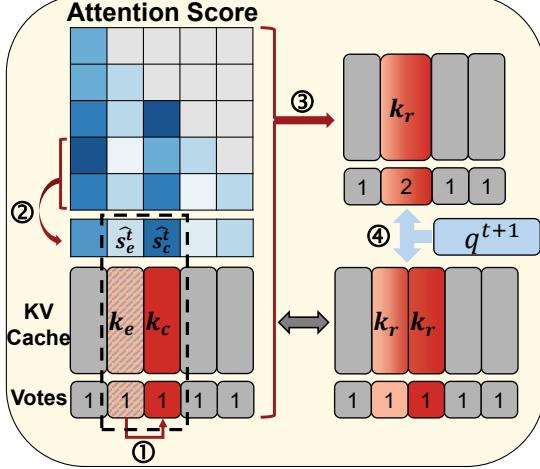
$$\hat{s}_i^t = \frac{1}{1 - \alpha^t} S^t, \quad S^t = \begin{cases} \sum_{k=t-w}^t (1 - \alpha) \alpha^{t-k} s^k, & t = L \\ \alpha S^{t-1} + (1 - \alpha) s^t, & t > L \end{cases} \quad (8)$$

Note that after the prefill stage, we compute EMA scores using a recent window of length w rather than the entire sequence to obtain a more accurate estimation (Li et al. 2024; Yang et al. 2024a). We find that this method outperforms mainstream approaches, such as cumulative attention and sliding window averaging, in predicting attention scores. Building on this, replacing all score s_i^t in Equation 7 with our EMA scores \hat{s}_i^t from Equation 8 successfully achieves the extension to multi-step generation. Consequently, the future output perturbation becomes estimable and controllable. We present the following theorem and lemma(proof in Appendix A.4):

Theorem 5. *For the t' -th step, let $\left|1 - \frac{\hat{s}_i^{t'}}{s_i^{t'}}$ $\right| \leq \epsilon, \epsilon < 1$, the output perturbation satisfies $\Theta_{t'} < \frac{2\epsilon(1+\epsilon)\gamma}{(1-\epsilon)^2}$, provided that $\|v_i - v_j\| \leq \gamma, \forall i \in [t'], j \in \{e, c\}$.*

Lemma 6. *As the prediction error ϵ decreases and the merged candidates become increasingly similar, the output perturbation reduces to zero. That is, when either $\epsilon = 0$ or $(k_e, v_e) = (k_c, v_c)$, we have: $\Theta_{t'} = 0$.*

Similarity-driven merging. Lemma 6 shows that output perturbation decreases as prediction error ϵ reduces, and closer merging objects result in lower perturbation. Clearly, if the merged KV pairs are identical, retaining one pair and setting its Electoral Votes to 2 introduces no error in subsequent computations. This provides a theoretical justification for prior merging strategies favoring high-similarity KV pairs (Wan et al. 2024b; Wang et al. 2024). Following this, we merge each evicted KV pair with the retained one having the highest cosine similarity of keys, using a predefined threshold T to determine whether merging should occur, avoiding



Algorithm 1 KeepKV Merging at t -th Step

```

1: Input: Attention scores  $s^t$ , EMA scores  $S^{t-1}$ , KV cache
2: Let  $K_e$  denote the to-be-evicted cache
3: Let  $K_c$  denote the retained cache
4:  $U = \text{cosineSimilarity}(K_e, K_c)$ 
5: For each  $k_e \in K_e$ , select  $k_c = \text{Argmax}_{k_c \in K_c}(U^e)$ ,  $U_{e,c} > T$ 
6:  $\hat{s}^t = \text{updateEMA}(S^{t-1}, s^t)$  ▷ Eq. (8)
7: Merge:
8:  $k_r = \frac{\ln((p_e \hat{s}_e^t + p_c \hat{s}_c^t) / (p_e + p_c))}{p_e \hat{s}_e^t \ln \hat{s}_e^t + p_c \hat{s}_c^t \ln \hat{s}_c^t} (p_e \hat{s}_e^t k_e + p_c \hat{s}_c^t k_c)$ 
9:  $v_r = \frac{1}{p_e \hat{s}_e^t + p_c \hat{s}_c^t} (p_e \hat{s}_e^t v_e + p_c \hat{s}_c^t v_c)$ 
10:  $p_r = p_e + p_c$ 
11: Discard  $(k_e, v_e)$ ,  $(k_c, v_c)$  and insert  $(k_r, v_r)$  into KV cache
12: Output: Updated KV cache

```

Figure 3: Illustrative example of KeepKV. (0) (k_e, v_e) is selected for eviction by specific compression method. (1) The retained KV with the highest cosine similarity, (k_c, v_c) , is selected. (2) EMA attention scores are updated. (3) ZIP-Merging is performed. (4) Consequently, with the Electoral Votes, the compressed KV can preserve the influence of the original KV in attention computations.

the overhead of dynamic adjustments like D2O (Wan et al. 2024b). Furthermore, we observe that, during the prefill stage, reversing the conventional order—by first merging based on key similarity and then applying the eviction strategy, instead of merging after eviction as commonly done—can improve generation quality.

We present the workflow of KeepKV in Figure 3. Notably, KeepKV imposes no specific constraints on cache allocation or token selection strategies. It can directly integrate with common token selection methods by designating the merging pairs based on their eviction and retention sets, and it is also compatible with various cache allocation strategies. Thus, KeepKV demonstrates strong adaptability and can be combined with a range of mainstream cache compression methods, significantly enhancing both compression capability and generation quality.

4 Experiment

4.1 Experiment Settings

Tasks We evaluate KeepKV on datasets with standard and extended context lengths, covering question-answering, summarization, and synthetic tasks. Specifically, for question-answering, we utilize MathQA (Amini et al. 2019), OpenBookQA (Mihaylov et al. 2018) and other tasks from the lm-eval-harness framework (Gao et al. 2024). For summarization, we employ the XSUM(Narayan, Cohen, and Lapata 2018) and CNN/DailyMail(Nallapati et al. 2016) tasks provided by the HELM framework. To assess performance on long-context tasks, we adopt LongBench(Bai et al. 2024), which effectively examines the algorithm’s compression capabilities across diverse subtasks, including single-document QA, multi-document QA and synthetic tasks.

Models and baselines. Our evaluation is based on several representative LLMs, including Llama-2 (Touvron et al. 2023b), Llama-3 (Grattafiori, Dubey et al. 2024), and Mistral

(Jiang et al. 2023). We compare our method against multiple baseline approaches: representative cache eviction methods such as Streaming (Xiao et al. 2024b), H2O (Zhang et al. 2023) and PyramidInfer (Yang et al. 2024a), and prominent cache merging methods including CaM (Zhang et al. 2024) and D2O (Wan et al. 2024b). For all baselines, we follow the authors’ released implementations and hyperparameter settings in their papers.

Implementation. In our main experiments, we set the merging threshold T to 0.8. For token selection and cache allocation, we follow the strategy recommended by PyramidInfer (Yang et al. 2024a), which allocates fixed cache budgets, making it simple and efficient. And it is sufficient to demonstrate the advantages of our algorithm. In contrast, D2O (Wan et al. 2024b) applies dynamic allocation based on extra computation after prefill phase for each sequence. We implement KeepKV using the Hugging Face Transformers (Wolf et al. 2020) and experiments are by default conducted on NVIDIA A100 80GB GPUs.

4.2 Accuracy on KV Cache Compression Ratios

In Figure 4, we benchmark KeepKV on both the lm-eval-harness and HELM frameworks, comparing the fully cached KV version against multiple KV cache compression methods, including our proposed KeepKV. The x-axis represents the compression ratio, defined as the ratio between the compressed KV cache budget and the prompt length L . The results demonstrate that KeepKV consistently outperforms all other compression methods across various compression ratios. Particularly at extremely low compression rates, KeepKV achieves significantly better performance, highlighting its superior compression capability to retain maximal information within highly constrained memory budgets while effectively minimizing output perturbations introduced by compression.

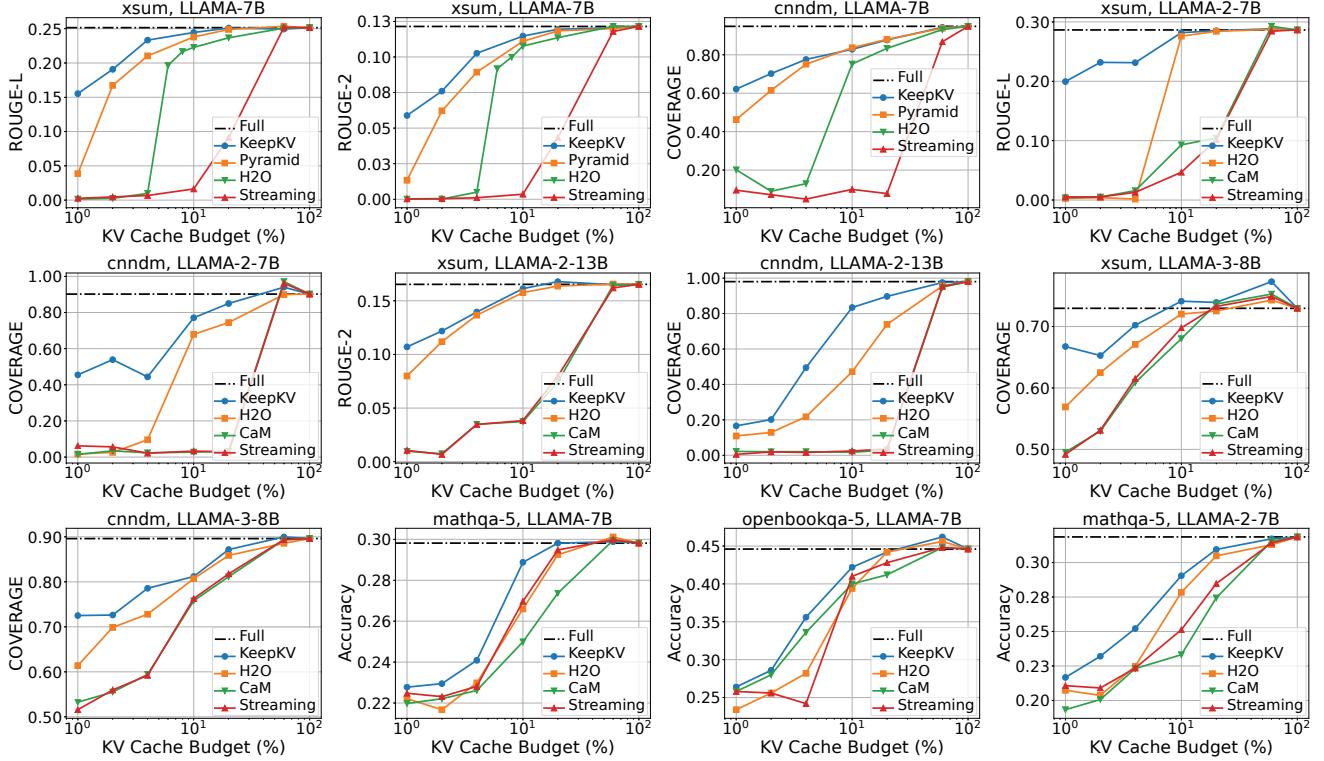


Figure 4: Performance of KeepKV and other methods for LLama backbones on HELM and LM-Eval evaluations.

Methods	Batch Size	Throughput (tokens/s)
Full cache	2	116.54
H2O	8	317.33
D2O	8	214.8
KeepKV	8	255.99

Table 1: Throughput comparison of KeepKV and other methods (4k context, 20% compression ratio).

4.3 Accuracy on Long-context Tasks

We evaluate KeepKV on the LongBench across Llama and Mistral model families, including Llama-2-7B, Llama-2-13B, Llama-3-8B and Mistral-7B, as shown in Table 2. The evaluation tasks include Single-Document QA, Multi-Document QA, Summarization, Synthetic, and Code. The results indicate that KeepKV achieves performance closer to the full-cache baseline on most tasks, maintaining high generation quality despite limited cache availability. Notably, KeepKV significantly outperforms eviction-based methods, such as Local Window, StreamingLLM (Xiao et al. 2024b), and H2O (Zhang et al. 2023). Furthermore, KeepKV also surpasses existing KV-cache merging methods, like CaM (Zhang et al. 2024) and D2O (Wan et al. 2024b), underscoring the effectiveness of our carefully designed merging strategy in enhancing output accuracy.

4.4 Throughput Analysis

Our experiments demonstrate that KeepKV significantly enhances the inference throughput of the model by efficiently compressing the KV Cache, as illustrated in Table 1. We conducted experiments on the Llama-2-7B model using an A100-80G GPU, with tasks derived from the LongBench evaluation framework. The experimental results indicate that various compression techniques improve throughput by reducing cache size and increasing batch size. Compared to the original full-cache method, KeepKV achieved over 2× increase in throughput. It is noteworthy that, due to the additional computations, the throughput per request of merging methods is typically lower than that of classical eviction methods, such as H2O (Zhang et al. 2023). Nonetheless, KeepKV achieves higher throughput than the state-of-the-art (SOTA) merging-based algorithm, D2O (Wan et al. 2024b). This advantage arises because D2O computes attention distribution variance for real-time cache allocation, whereas our method adopts a fixed cache allocation strategy to emphasize the generalizability of KeepKV.

4.5 Ablation Study

To evaluate the generalizability of KeepKV, we conducted ablation experiments combining KeepKV with existing state-of-the-art eviction methods. Since KeepKV does not impose specific requirements on cache allocation or eviction/preservation strategies, it can be directly integrated with commonly used eviction/preservation policies. This only requires setting the merged parties as the eviction and preservation sets deter-

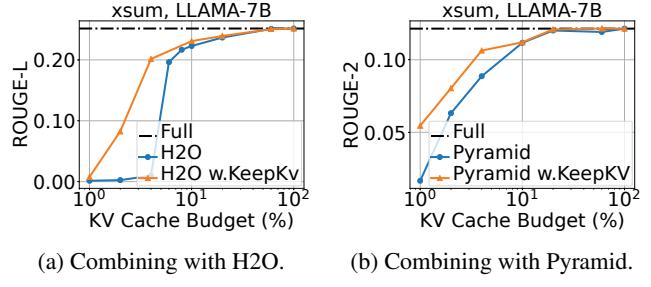
Methods	Single-Doc QA			Multi-Doc QA			Summarization			Synthetic		Code	
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P
Llama-2-7B													
Full Model	15.8	9.39	22.09	8.56	10.85	4.3	65.0	89.64	34.16	1.0	8.29	66.77	60.1
Local Window	2.22	9.29	1.83	5.14	7.18	1.02	17.5	4.07	3.17	1.5	2.58	16.31	15.35
StreamingLLM	11.81	5.18	19.26	7.07	10.48	3.71	55.5	87.31	31.84	1.5	4.29	63.79	56.07
H2O	16.54	7.57	20.61	7.68	9.28	4.09	64.0	87.98	33.62	1.34	9.14	65.34	58.49
CaM	11.79	5.1	19.12	7.26	10.48	3.64	56.0	87.31	31.85	1.5	4.29	63.66	55.98
D2O	16.04	6.54	19.48	8.14	10.12	4.62	63.5	88.39	34.1	1.39	7.54	65.8	59.44
Ours	17.32	7.48	22.2	8.51	9.72	4.65	60.5	88.87	33.2	2.23	8.45	65.9	56.36
Llama-2-13B													
Full Model	12.64	8.61	19.82	9.1	10.98	5.8	69.5	87.04	41.89	2.0	6.03	67.08	57.53
Local	4.95	5.11	3.82	7.05	9.87	3.42	19.0	7.83	2.63	1.17	6.51	16.7	14.65
StreamingLLM	5.04	5.75	12.24	9.4	10.47	4.71	57.0	82.48	37.21	1.5	5.04	61.47	50.84
H2O	13.83	6.41	15.52	9.04	9.55	5.53	66.0	86.08	40.2	2.88	7.37	64.52	55.46
CaM	5.16	5.95	12.31	9.19	10.52	4.66	57.0	82.48	37.28	2.5	5.25	61.75	50.71
D2O	12.76	6.53	14.87	8.59	10.34	5.75	66.5	86.52	40.52	2.0	6.99	65.23	55.84
Ours	12.09	6.89	17.81	9.49	10.54	5.79	66.8	82.72	41.35	1.75	7.55	64.81	56.29
Llama-3-8B													
Full Model	14.34	13.68	21.7	9.42	10.75	6.99	72	90.7	45.13	3.74	6.72	70.54	66.04
Local	2.14	6.69	5.17	6.16	5.0	2.42	34.25	30.5	10.66	2.36	2.0	28.91	24.52
StreamingLLM	10.43	7.84	13.85	9.18	10.44	5.47	61.0	90.37	44.35	2.6	10.5	68.49	63.94
H2O	13.73	10.02	17.2	9.31	10.62	6.42	63.3	90.44	45.02	3.29	7.56	68.95	63.84
CaM	10.43	7.83	13.89	9.11	10.37	5.47	61.0	90.37	44.31	3.16	10.5	68.59	64.04
D2O	13.5	8.86	17.21	9.16	10.52	6.35	65.5	90.52	44.64	3.44	5.8	68.49	64.84
Ours	12.76	10.63	18.57	9.37	10.72	6.53	64.5	90.33	45.2	3.54	7.16	69.05	65.68
Mistral-7B													
Full Model	22.92	39.74	51.46	43.28	39.46	25.59	74.0	88.64	46.97	4.0	63.5	61.42	58.72
Local	16.89	16.92	21.11	23.33	22.49	10.23	58.5	81.29	36.3	2.1	7.71	41.1	47.88
StreamingLLM	16.76	17.28	21.41	24.16	22.54	10.72	60.3	82.21	37.43	2.14	7.67	51.19	47.94
H2O	18.06	16.75	22.28	24.77	21.68	8.86	61.0	83.03	30.34	2.15	5.76	56.5	49.88
CaM	16.46	17.26	21.4	25.66	22.54	10.72	59.17	82.21	37.33	2.14	7.67	51.01	47.89
D2O	18.58	15.92	21.71	26.41	21.68	9.07	61.5	83.12	39.5	2.18	7.3	57.51	50.59
Ours	18.16	17.95	22.93	26.56	23.18	9.42	62	83.47	39.7	2.19	7.26	58.9	50.71

Table 2: Performance evaluation of KeepKV on various models in LongBench benchmarks (20% compression ratio).

mined by their respective algorithms; it can also be applied with various cache allocation strategies, simply by modifying the cache configurations between layers and attention heads. As shown in Figure 5, we combined KeepKV with existing mainstream eviction methods, H2O(Zhang et al. 2023) and PyramidInfer(Yang et al. 2024a), using the HELM evaluation framework. The results demonstrate that, with KeepKV incorporated, the methods outperform the original ones across all compression ratios. This proves that our algorithm is highly scalable and versatile, capable of being integrated with various eviction schemes to enhance their compression efficiency and generation quality.

5 Conclusion

In this paper, we conduct a comprehensive analysis of the impact of KV cache compression on attention computation and propose KeepKV, which introduces the Electoral Votes mechanism and Zero Inference-Perturbation Merging to adaptively and dynamically merge the KV cache while minimizing output disturbance. KeepKV effectively preserves more information within limited memory, significantly mitigating



(a) Combining with H2O.

(b) Combining with Pyramid.

Figure 5: Accuracy experiments combining KeepKV with existing eviction methods.

the adverse effects of KV cache compression on generation quality. Our experiments demonstrate that KeepKV achieves performance closest to that of the full cache across various compression ratios. It also excels in both standard and long-context tasks. We believe KeepKV provides a novel perspective and a powerful tool for advancing KV cache compression methods, laying the foundation for efficient LLM inference.

Acknowledgements

We are grateful to Chenhong He, Ruijie Miao, Yuhua Wu and Yanhu Wang from Peking University for their insightful discussions and helpful suggestions throughout the development of this research. We thank ByteDance Ltd. for providing technical support during the internship period. This work was supported by the National Key Research and Development Program of China under Grant No. 2024YFB2906603, by the Beijing Natural Science Foundation (Grant No. QY25123), and in part by the National Natural Science Foundation of China (NSFC) (624B2005).

References

- Amini, A.; Gabriel, S.; Lin, P.; Koncel-Kedziorski, R.; Choi, Y.; and Hajishirzi, H. 2019. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. arXiv:1905.13319.
- Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; Dong, Y.; Tang, J.; and Li, J. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. arXiv:2308.14508.
- Busbridge, D.; Ramapuram, J.; Ablin, P.; Likhomanenko, T.; Dhekane, E. G.; Suau, X.; and Webb, R. 2023. How to Scale Your EMA. arXiv:2307.13813.
- Cai, Z.; Zhang, Y.; Gao, B.; Liu, Y.; Liu, T.; Lu, K.; Xiong, W.; Dong, Y.; Chang, B.; Hu, J.; and Xiao, W. 2024. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069.
- Chen, Z.; Sadhukhan, R.; Ye, Z.; Zhou, Y.; Zhang, J.; Nolte, N.; Tian, Y.; Douze, M.; Bottou, L.; Jia, Z.; and Chen, B. 2024. MagicPIG: LSH Sampling for Efficient LLM Generation. arXiv:2410.16179.
- Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J. G.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Annual Meeting of the Association for Computational Linguistics*.
- Dong, S.; Cheng, W.; Qin, J.; and Wang, W. 2024. QAQ: Quality Adaptive Quantization for LLM KV Cache. arXiv:2403.04643.
- Feng, Y.; Lv, J.; Cao, Y.; Xie, X.; and Zhou, S. K. 2024. AdaKV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. arXiv:2407.11550.
- Fu, Y.; Cai, Z.; Asi, A.; Xiong, W.; Dong, Y.; and Xiao, W. 2024. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. arXiv:2410.19258.
- Gao, L.; Tow, J.; Abbasi, B.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; Le Noac'h, A.; Li, H.; McDonell, K.; Muennighoff, N.; Ociepa, C.; Phang, J.; Reynolds, L.; Schoelkopf, H.; Skowron, A.; Sutawika, L.; Tang, E.; Thite, A.; Wang, B.; Wang, K.; and Zou, A. 2024. A framework for few-shot language model evaluation.
- Grattafiori, A.; Dubey, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.
- Gu, X.; Pang, T.; Du, C.; Liu, Q.; Zhang, F.; Du, C.; Wang, Y.; and Lin, M. 2025. When Attention Sink Emerges in Language Models: An Empirical View. arXiv:2410.10781.
- Han, C.; Wang, Q.; Peng, H.; Xiong, W.; Chen, Y.; Ji, H.; and Wang, S. 2024. LM-Infinite: Zero-Shot Extreme Length Generalization for Large Language Models. arXiv:2308.16137.
- Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M. W.; Shao, Y. S.; Keutzer, K.; and Gholami, A. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079.
- Hunter, J. S. 1986. The exponentially weighted moving average. *Journal of quality technology*, 18(4): 203–210.
- Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Scao, T. L.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2023. Mistral 7B. arXiv:2310.06825.
- Kang, H.; Zhang, Q.; Kundu, S.; Jeong, G.; Liu, Z.; Krishna, T.; and Zhao, T. 2024. GEAR: An Efficient KV Cache Compression Recipe for Near-Lossless Generative Inference of LLM. arXiv:2403.05527.
- Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2307.08999.
- Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. arXiv:2404.14469.
- Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; and Hu, X. 2024. ScissorHands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. arXiv:2402.02750.
- Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. arXiv:1809.02789.
- Nallapati, R.; Zhou, B.; dos santos, C. N.; Gulcehre, C.; and Xiang, B. 2016. Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond. arXiv:1602.06023.
- Narayan, S.; Cohen, S. B.; and Lapata, M. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. arXiv:1808.08745.
- Nawrot, P.; Łaćucki, A.; Chochowski, M.; Tarjan, D.; and Ponti, E. M. 2024. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. arXiv:2403.09636.
- OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2024. GPT-4 Technical Report. arXiv:2303.08774.
- Rae, J. W.; Potapenko, A.; Jayakumar, S. M.; and Lillicrap, T. P. 2019. Compressive Transformers for Long-Range Sequence Modelling. *ArXiv*, abs/1911.05507.
- Reid, S.; and Zhu, K. 2024. On the Efficacy of Eviction Policy for Key-Value Constrained Generative Language Model Inference. arXiv:2402.06262.

- Rozière, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X. E.; Adi, Y.; Liu, J.; Sauvestre, R.; Remez, T.; Rapin, J.; Kozhevnikov, A.; Evtimov, I.; Bitton, J.; Bhatt, M.; Ferrer, C. C.; Grattafiori, A.; Xiong, W.; Défossez, A.; Copet, J.; Azhar, F.; Touvron, H.; Martin, L.; Usunier, N.; Scialom, T.; and Synnaeve, G. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950.
- Shi, L.; Zhang, H.; Yao, Y.; Li, Z.; and Zhao, H. 2024. Keep the Cost Down: A Review on Methods to Optimize LLM’s KV-Cache Consumption. arXiv:2407.18003.
- Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023a. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.
- Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.
- Vaswani, A.; Shazeer, N. M.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Neural Information Processing Systems*.
- Wan, Z.; Wang, X.; Liu, C.; Alam, S.; Zheng, Y.; Liu, J.; Qu, Z.; Yan, S.; Zhu, Y.; Zhang, Q.; Chowdhury, M.; and Zhang, M. 2024a. Efficient Large Language Models: A Survey. arXiv:2312.03863.
- Wan, Z.; Wu, X.; Zhang, Y.; Xin, Y.; Tao, C.; Zhu, Z.; Wang, X.; Luo, S.; Xiong, J.; and Zhang, M. 2024b. D2O: Dynamic Discriminative Operations for Efficient Generative Inference of Large Language Models. arXiv:2406.13035.
- Wang, Z.; Jin, B.; Yu, Z.; and Zhang, M. 2024. Model Tells You Where to Merge: Adaptive KV Cache Merging for LLMs on Long-Context Tasks. arXiv:2407.08454.
- Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2020. HuggingFace’s Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771.
- Xiao, G.; Tang, J.; Zuo, J.; Guo, J.; Yang, S.; Tang, H.; Fu, Y.; and Han, S. 2024a. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. arXiv:2410.10819.
- Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2024b. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453.
- Yang, D.; Han, X.; Gao, Y.; Hu, Y.; Zhang, S.; and Zhao, H. 2024a. PyramidInfer: Pyramid KV Cache Compression for High-Throughput LLM Inference. arXiv:2406.02069.
- Yang, J. Y.; Kim, B.; Bae, J.; Kwon, B.; Park, G.; Yang, E.; Kwon, S. J.; and Lee, D. 2024b. No Token Left Behind: Reliable KV Cache Compression via Importance-Aware Mixed Precision Quantization. arXiv:2402.18096.
- Zhang, Y.; Du, Y.; Luo, G.; Zhong, Y.; Zhang, Z.; Liu, S.; and Ji, R. 2024. CaM: Cache Merging for Memory-efficient LLMs Inference. In *Forty-first International Conference on Machine Learning*.
- Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36: 34661–34710.

A Theoretical Analysis

Recently, many studies have analyzed KV cache compression strategies in LLM inference from a theoretical perspective (Zhang et al. 2023; Liu et al. 2024; Li et al. 2024; Yang et al. 2024a; Zhang et al. 2024; Wan et al. 2024b; Wang et al. 2024; Gu et al. 2025). Overall, the primary objective of most existing works can be summarized as minimizing the impact of compression on the output. For instance, existing eviction-based methods and cache allocation strategies (Zhang et al. 2023; Liu et al. 2024; Reid and Zhu 2024; Li et al. 2024; Yang et al. 2024a; Cai et al. 2024; Feng et al. 2024; Fu et al. 2024; Xiao et al. 2024a) all aim to maximize the retention of essential information within limited memory by evicting less important tokens or reducing cache allocation in non-critical heads and layers based on empirical observations of attention distributions. However, eviction inevitably leads to irreversible information loss, which has motivated the development of KV cache merging methods (Zhang et al. 2024; Wan et al. 2024b; Wang et al. 2024; Nawrot et al. 2024). Despite this, key challenges such as the selection of merging candidates and the assignment of merging weights remain largely unexplored, with a lack of systematic theoretical foundations. In this work, we introduce a novel perspective distinct from prior approaches. We formulate the problem as eliminating output perturbation and derive a novel merging method by analyzing the attention computation process. First, we introduce Electoral Votes mechanism, making the elimination of output perturbation feasible. Then, we derive a merging computation formula to eliminate perturbation at the current step. Finally, we extend this framework to multi-step generation, providing a theoretical guarantee for output perturbation and offering a reasonable explanation for mainstream similarity-based merging candidates selection methods.

Specifically, we first demonstrate the unavoidable output perturbation caused by KV cache eviction. Next, we discuss the attention sag issue in existing KV cache merging methods and provide a formal proof. Then, we present the derivation process of our KeepKV merging method. Finally, we provide a theoretical guarantee for the output perturbation of KeepKV, including proofs for the main theorem and its associated lemma. The symbolic representation of the attention computation process remains consistent with the one introduced in the methodology section.

A.1 Perturbation in KV Cache Eviction

Eviction methods discard KV pairs deemed unimportant. We denote the first generation step in the decoding phase as the $(L + 1)$ -th generation step, where L represents the prompt length. And for a positive integer n , let $[n] := \{1, 2, \dots, n\}$. At t -th generation step, let $K_e = \{e_1 e_2, \dots, e_m\}$, $m \in [t]$

denoted the index of to-be-evicted cache. Based on Equation 3, the output after eviction (o'_t) is:

$$o'_t = \sum_{i=1, i \notin K_e}^t A_i'^t v_i, \quad A_i'^t = \frac{s_i^t}{\sum_{i=1, i \notin K_e}^t s_i^t}. \quad (9)$$

By transforming o'_t towards o_t , we obtain:

$$\begin{aligned} o'_t &= \frac{\sum_{i=1}^t s_i^t}{\sum_{i=1}^t s_i^t - \sum_{j \in K_e}^t s_j^t} * \frac{\sum_{i=1}^t s_i^t v_i - \sum_{j \in K_e}^t s_j^t v_j}{\sum_{i=1}^t s_i^t} \\ &= \frac{\sum_{i=1}^t s_i^t}{\sum_{i=1}^t s_i^t - \sum_{j \in K_e}^t s_j^t} \left(o_t - \frac{\sum_{j \in K_e}^t s_j^t v_j}{\sum_{i=1}^t s_i^t} \right) \\ &= \frac{\sum_{i=1}^t A_i^t}{\sum_{i=1}^t A_i^t - \sum_{j \in K_e}^t A_j^t} \left(o_t - \frac{\sum_{j \in K_e}^t A_j^t v_j}{\sum_{i=1}^t A_i^t} \right) \\ &= \frac{1}{1 - \sum_{j \in K_e}^t A_j^t} \left(o_t - \sum_{j \in K_e}^t A_j^t v_j \right). \end{aligned} \quad (10)$$

Equation 10 indicates that the difference between o'_t and o_t decreases as the attention score of the evicted KV ($A_j^t, j \in K_e$) diminishes. When the total score of K_e becomes negligible, the output perturbation at the current step approaches zero. This formally explains why eviction methods generally prioritize discarding KV pairs with lower attention scores.

However, existing studies (Chen et al. 2024) have shown that attention can be relatively dispersed in certain tasks, meaning that evicting even a small number of tokens can have a non-negligible impact. Furthermore, as the compression ratio increases, evicted tokens will account for a significant portion of the attention scores, exacerbating the degradation of generation quality.

A.2 Attention Sag in KV Cache Merging

Merging methods integrate less important KV into others rather than discarding them directly. Specifically, mainstream studies select, for each KV pair to be evicted, a merging target among the preserved KVs, allowing many-to-one merges. Typically, weighted merging rather than direct averaging is used, with weights satisfying a normalization constraint, i.e., the merged vectors are obtained via convex combinations. Formally, merging the evicted pairs $(k_j, v_j), j \in K_e$ into a preserved pair (k_r, v_r) yields a new KV pair (k_r, v_r) , defined as follows:

$$\begin{aligned} k_r &= w_c k_c + \sum_{j \in K_e} w_j k_j, \\ v_r &= w_c v_c + \sum_{j \in K_e} w_j v_j, \\ \text{s.t. } w_c + \sum_{j \in K_e} w_j &= 1 \end{aligned} \quad (11)$$

Let $K'_e = K_e \cup \{c\}$, representing the index of the original KVs before merging. For instance, the weight w_j in D2O (Wan et al. 2024b) is computed based on the cosine similarity between key vectors, whereas for KVmerger (Wang et al. 2024), it is calculated based on the Gaussian Kernel value. Formally, these are represented as follows:

$$\begin{aligned} w_{j_{D2O}} &= \frac{\exp(\cos \theta_{k_j, k_c})}{\sum_{j \in K'_e} \exp(\cos \theta_{k_j, k_c})}, \\ w_{j_{KVMerger}} &= \frac{\exp\left(-\frac{\|k_j - k_c\|^2}{2\sigma^2}\right)}{\sum_{j \in K'_e} \exp\left(-\frac{\|k_j - k_c\|^2}{2\sigma^2}\right)}. \end{aligned} \quad (12)$$

However, the widely adopted convex combination approach also introduces output disturbances, as stated in the following theorem:

Theorem 7 (Formal version of Theorem 2). *The merging method indicated by Equation 11 causes the attention score of the merged KV to become less than the sum of attention scores from the original multiple KVs merged into it, independently of the specific weighting scheme. Formally, this implies: $A_r'^t < \sum_{j \in K'_e} A_j^t$, ultimately leading to: $\|o'_t - o_t\| > 0$.*

Proof. The attention score and output after merging can be expressed as:

$$o'_t = \sum_{i=1, i \notin K'_e}^t A_i'^t v_i + A_r'^t v_r = \frac{\sum_{i=1, i \notin K'_e}^t s_i^t v_i + s_r^t v_r}{\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t}. \quad (13)$$

First, we compare the denominators of A^t and A'^t , formally proving $\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t < \sum_{i=1}^t s_i^t$:

$$\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t = \sum_{i=1}^t s_i^t + s_r^t - \sum_{i \in K'_e} s_i^t = e^{\frac{q^t k_r}{\sqrt{d}}} - \sum_{i \in K'_e} e^{\frac{q^t k_i}{\sqrt{d}}} \quad (14)$$

Substituting Equation 11, and applying the Weighted AM–GM Inequality, we have:

$$\begin{aligned} e^{\frac{q^t k_r}{\sqrt{d}}} - \sum_{i \in K'_e} e^{\frac{q^t k_i}{\sqrt{d}}} &= e^{\frac{q^t (\sum_{i \in K'_e} w_i k_i)}{\sqrt{d}}} - \sum_{i \in K'_e} e^{\frac{q^t k_i}{\sqrt{d}}} \\ &= \prod_{i \in K'_e} (e^{\frac{q^t k_i}{\sqrt{d}}})^{w_i} - \sum_{i \in K'_e} e^{\frac{q^t k_i}{\sqrt{d}}} \\ &\leq \sum_{i \in K'_e} w_i e^{\frac{q^t k_i}{\sqrt{d}}} - \sum_{i \in K'_e} e^{\frac{q^t k_i}{\sqrt{d}}} < 0 \end{aligned} \quad (15)$$

Thus,

$$\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t = \sum_{i=1}^t s_i^t + (s_r^t - \sum_{i \in K'_e} s_i^t) < \sum_{i=1}^t s_i^t \quad (16)$$

Since the sum of the normalized attention scores equals one, and given that $\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t < \sum_{i=1}^t s_i^t$, we obtain:

$$\begin{aligned} A'^t_r &= 1 - \frac{\sum_{i=1, i \notin K'_e}^t s_i^t}{\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t} \\ &< 1 - \frac{\sum_{i=1, i \notin K'_e}^t s_i^t}{\sum_{i=1}^t s_i^t} = \sum_{i \in K'_e} A_i^t \end{aligned} \quad (17)$$

Similarly, we can derive:

$$A'^t_j = \frac{s_j^t}{\sum_{i=1, i \notin K'_e}^t s_i^t + s_r^t} > \frac{s_j^t}{\sum_{i=1}^t s_i^t} = A_j^t, \quad j \neq r \quad (18)$$

Finally, the output perturbation can be represented as:

$$\begin{aligned} \|o'_t - o_t\| &= \left\| \left(\sum_{i=1, i \notin K'_e}^t A'_i v_i + A'^t_r v_r \right) - \sum_{i=1}^t A_i^t v_i \right\| \\ &= \left\| \sum_{i=1, i \notin K'_e}^t (A'_i - A_i^t) v_i + (A'^t_r v_r - \sum_{i \in K'_e} A_i^t v_i) \right\| \\ &= \left\| \sum_{i=1, i \notin K'_e}^t (A'_i - A_i^t) v_i + \sum_{i \in K'_e} (w_i A'^t_r - A_i^t) v_i \right\| \end{aligned} \quad (19)$$

In the above expression, all vector coefficients are nonzero. Moreover, due to the high dimensionality and sparsity of the KV cache (Wang et al. 2024; Gu et al. 2025), the vectors are almost linearly independent. In practical inference scenarios, it is impossible for them to form a zero vector through linear combination. Consequently, we have: $\|o'_t - o_t\| > 0$. \square

We term this phenomenon as **Attention Sag**, indicating that improper merging methods result in a reduced attention score for the newly merged vector, while attention scores of unmerged KVs relatively increase. This leads to output disturbances and ultimately degrades generation quality.

A.3 KeepKV Merging Method

In the main text, we introduced the concept of merging count via the Electoral Votes mechanism, aiming for a KV pair with vote count p_i to be equivalent, in attention computation, to p_i independent occurrences of this KV. Moreover, the vote count of the merged KV equals the sum of vote counts before merging. Formally, the outputs before (o_t) and after merging (o'_t) can be expressed as follows:

$$\begin{aligned} o_t &= \frac{\sum_{i=1}^t p_i s_i^t v_i}{\sum_{i=1}^t p_i s_i^t}, \\ o'_t &= \frac{\sum_{i=1, i \notin K'_e}^t p_i s_i^t v_i + p_r s_r^t v_r}{\sum_{i=1, i \notin K'_e}^t p_i s_i^t + p_r s_r^t}, \\ p_r &= \sum_{i \in K'_e} p_i. \end{aligned} \quad (20)$$

Next, we demonstrate how our new merging approach can be derived naturally from the objective of eliminating output disturbances, which consequently serves as a direct proof for Theorem 3.

Based on Equation 20, setting $\|o'_t - o_t\| = 0$, we obtain:

$$\begin{aligned} \sum_{i=1}^t p_i s_i^t v_i &= \sum_{i=1, i \notin K'_e}^t p_i s_i^t v_i + p_r s_r^t v_r, \\ \sum_{i=1}^t p_i s_i^t &= \sum_{i=1, i \notin K'_e}^t p_i s_i^t + p_r s_r^t. \end{aligned} \quad (21)$$

which implies:

$$\sum_{i \in K'_e} p_i s_i^t v_i = p_r s_r^t v_r, \quad \sum_{i \in K'_e} p_i s_i^t = p_r s_r^t \quad (22)$$

Dividing the two expressions above, we obtain the expression of v_r :

$$v_r = \frac{\sum_{i \in K'_e} p_i s_i^t v_i}{\sum_{i \in K'_e} p_i s_i^t} \quad (23)$$

Similarly, let $k_r = C(\sum_{i \in K'_e} p_i s_i^t k_i)$, substituting this into $\sum_{i \in K'_e} p_i s_i^t = p_r s_r^t$ from Equation 22 and solving, we obtain:

$$C = \frac{\ln \frac{\sum_{i \in K'_e} p_i s_i^t}{\sum_{i \in K'_e} p_i}}{\sum_{i \in K'_e} p_i s_i^t \ln s_i^t} \quad (24)$$

Finally, we derive the merging expression:

$$\begin{aligned} k_r &= \frac{\left(\sum_{i \in K'_e} p_i s_i^t k_i \right) \ln \left(\frac{\sum_{i \in K'_e} p_i s_i^t}{\sum_{i \in K'_e} p_i} \right)}{\sum_{i \in K'_e} p_i s_i^t \ln s_i^t}, \\ v_r &= \frac{\sum_{i \in K'_e} p_i s_i^t v_i}{\sum_{i \in K'_e} p_i s_i^t}, \quad p_r = \sum_{i \in K'_e} p_i. \end{aligned} \quad (25)$$

Consequently, merging in this manner eliminates the output disturbance in the t -step, satisfying: $\|o'_t - o_t\| = 0$. By setting the merging candidates $K'_e = \{e\} \cup \{c\}$, we obtain Theorem 3.

A.4 Error Bound Analysis

After extending KeepKV to multi-step generation, for t' -step, all s_i^t terms in Equation 25 are replaced with $\hat{s}_i^{t'}$, which represents our estimation of future attention score trends obtained through a certain method. In this case, the merging expressions become:

$$\begin{aligned} k_r &= \frac{\left(\sum_{i \in K'_e} p_i \hat{s}_i^{t'} k_i \right) \ln \left(\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i} \right)}{\sum_{i \in K'_e} p_i \hat{s}_i^{t'} \ln \hat{s}_i^{t'}}, \\ v_r &= \frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'} v_i}{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}, \quad p_r = \sum_{i \in K'_e} p_i. \end{aligned} \quad (26)$$

and they satisfy:

$$\sum_{i \in K'_e} p_i \hat{s}_i^{t'} v_i = p_r \hat{s}_r^t v_r, \quad \sum_{i \in K'_e} p_i \hat{s}_i^t = p_r \hat{s}_r^t \quad (27)$$

Then the perturbation at step s' , can be expressed as:

$$\begin{aligned} \Theta_{t'} &= \|o_{t'} - o'_{t'}\| \\ &= \left\| \frac{\sum_{i=1}^{t'} p_i s_i^{t'} v_i}{\sum_{i=1}^{t'} p_i s_i^{t'}} - \frac{\sum_{i=1, i \notin K'_e}^{t'} p_i s_i^{t'} v_i + p_r s_r^{t'} v_r}{\sum_{i=1, i \notin K'_e}^{t'} p_i s_i^{t'} + p_r s_r^{t'}} \right\| \\ &= \frac{\left\| \sum_{i=1}^{t'} p_i s_i^{t'} \left(\sum_{j \in K'_e} p_j s_j^{t'} (v_j - v_i) - p_r s_r^{t'} (v_r - v_i) \right) \right\|}{\left(\sum_{i=1}^{t'} p_i s_i^{t'} \right) \left(\sum_{i=1, i \notin K'_e}^{t'} p_i s_i^{t'} + p_r s_r^{t'} \right)} \end{aligned} \quad (28)$$

Substituting the expression for v_r from Equation 26 and $\sum_{j \in K'_e} p_j \hat{s}_j^{t'} = p_r \hat{s}_r^{t'}$ from Equation 27 into the above:

$$\begin{aligned} &\sum_{i=1}^{t'} p_i s_i^{t'} \left[\sum_{j \in K'_e} p_j \hat{s}_j^{t'} (v_j - v_i) - p_r s_r^{t'} \left(\frac{\sum_{k \in K'_e} p_k \hat{s}_k^{t'} v_k}{\sum_{k \in K'_e} p_k \hat{s}_k^{t'}} - v_i \right) \right] \\ &= \sum_{i=1}^{t'} p_i s_i^{t'} \sum_{j \in K'_e} p_j \hat{s}_j^{t'} \left(1 - \frac{s_r^{t'}}{\hat{s}_r^{t'}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \right) (v_j - v_i) \end{aligned} \quad (29)$$

Let $\left| 1 - \frac{s_r^{t'}}{s_i^{t'}} \right| \leq \epsilon, \epsilon < 1$, then $1 - \epsilon \leq \frac{s_i^{t'}}{s_r^{t'}} \leq 1 + \epsilon$, thus:

$$\left| 1 - \frac{s_r^{t'} \hat{s}_j^{t'}}{s_r^{t'} s_j^{t'}} \right| = \left| \frac{\frac{\hat{s}_r^{t'}}{s_r^{t'}} - \frac{s_j^{t'}}{s_r^{t'}}}{\frac{\hat{s}_r^{t'}}{s_r^{t'}}} \right| \leq \frac{2\epsilon}{1-\epsilon}, \quad j \in K'_e \quad (30)$$

Let $\|v_j - v_i\| \leq \gamma, \forall i \in [t'], j \in K'_e$, where γ represents the inherent variation in the input, which cannot be eliminated through algorithmic design. Then, applying the triangle inequality, we obtain:

$$\begin{aligned} &\left\| \sum_{i=1}^{t'} p_i s_i^{t'} \left[\sum_{j \in K'_e} p_j \hat{s}_j^{t'} \left(1 - \frac{s_r^{t'}}{s_r^{t'}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \right) (v_j - v_i) \right] \right\| \\ &\leq \sum_{i=1}^{t'} p_i s_i^{t'} \left(\sum_{j \in K'_e} p_j \hat{s}_j^{t'} \left| 1 - \frac{s_r^{t'}}{s_r^{t'}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \right| \cdot \|v_j - v_i\| \right) \\ &\leq \frac{2\epsilon\gamma}{1-\epsilon} \left(\sum_{i=1}^{t'} p_i s_i^{t'} \right) \left(\sum_{j \in K'_e} p_j \hat{s}_j^{t'} \right) \end{aligned} \quad (31)$$

Substituting this inequality into Equation 28, we obtain:

$$\begin{aligned} \Theta_{t'} &\leq \frac{2\epsilon\gamma}{1-\epsilon} \frac{(\sum_{i=1}^{t'} p_i s_i^{t'}) (\sum_{j \in K'_e} p_j \hat{s}_j^{t'})}{(\sum_{i=1}^{t'} p_i s_i^{t'}) (\sum_{i=1, i \notin K'_e}^{t'} p_i s_i^{t'} + p_r s_r^{t'})} \\ &= \frac{2\epsilon\gamma}{1-\epsilon} \frac{\sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{\sum_{i=1, i \notin K'_e}^{t'} p_i s_i^{t'} + p_r s_r^{t'}} \\ &< \frac{2\epsilon\gamma}{1-\epsilon} \frac{\sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{p_r s_r^{t'}} \end{aligned} \quad (32)$$

Due to Equation 27, we have $\frac{\sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{p_r s_r^{t'}} = 1$, then:

$$\frac{\sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{p_r s_r^{t'}} \leq \frac{\frac{1}{1-\epsilon} \sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{\frac{1}{1+\epsilon} p_r s_r^{t'}} = \frac{1+\epsilon}{1-\epsilon} \quad (33)$$

Thus,

$$\Theta_{t'} < \frac{2\epsilon\gamma}{1-\epsilon} \frac{\sum_{j \in K'_e} p_j \hat{s}_j^{t'}}{p_r s_r^{t'}} < \frac{2\epsilon(1+\epsilon)\gamma}{(1-\epsilon)^2} \quad (34)$$

Finally, we obtain the following theorem:

Theorem 8. For the t' -th step, let $\left| 1 - \frac{\hat{s}_i^{t'}}{s_i^{t'}} \right| \leq \epsilon, \epsilon < 1$, the output perturbation satisfies $\Theta_{t'} < \frac{2\epsilon(1+\epsilon)\gamma}{(1-\epsilon)^2}$, provided that $\|v_j - v_i\| \leq \gamma, \forall i \in [t'], j \in K'_e$.

Next, we prove the following lemma:

Lemma 9. As the prediction error ϵ decreases and the merged candidates become increasingly similar, the output disturbance reduces to zero. That is, when either $\epsilon = 0$ or $(k_i, v_i) = (k_j, v_j), \forall i, j \in K'_e$, we have: $\Theta_{t'} = 0$.

Proof. By Theorem 8, it is easy to obtain that when $\epsilon = 0$, $\Theta_{t'} < \frac{2\epsilon(1+\epsilon)\gamma}{(1-\epsilon)^2} = 0$. Next, we prove that when $(k_i, v_i) = (k_j, v_j), \forall i, j \in K'_e$, it also holds that $\Theta_{t'} = 0$. First, we further expand $(1 - \frac{s_r^{t'} \hat{s}_j^{t'}}{s_r^{t'} s_j^{t'}}), j \in K'_e$ in Equation 29 by applying Equation 27:

$$1 - \frac{s_r^{t'} \hat{s}_j^{t'}}{s_r^{t'} s_j^{t'}} = 1 - \frac{e^{\frac{q_{t'} k_r}{\sqrt{d}}}}{\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i}} \frac{\hat{s}_j^{t'}}{s_j^{t'}}, \quad j \in K'_e \quad (35)$$

Substituting the expression for k_r from Equation 26 into the above:

$$\begin{aligned} &1 - \frac{e^{\frac{q_{t'} k_r}{\sqrt{d}}}}{\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \\ &= 1 - \frac{\left(\prod_{i \in K'_e} (s_i^{t'})^{p_i \hat{s}_i^{t'}} \right)^{\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i}}}{\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \end{aligned} \quad (36)$$

When $(k_i, v_i) = (k_j, v_j), \forall i, j \in K'_e$, it follows that $\forall i \in K'_e, s_i^{t'} = s^{t'}, \hat{s}_i^{t'} = \hat{s}^{t'}$, thereby:

$$\begin{aligned}
& 1 - \frac{\left(\prod_{i \in K'_e} s_i^{t'} p_i \hat{s}_i^{t'} \right)^{\ln \left(\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i} \right)} \cdot \hat{s}_j^{t'}}{\frac{\sum_{i \in K'_e} p_i \hat{s}_i^{t'}}{\sum_{i \in K'_e} p_i}} \\
&= 1 - \frac{\left(\prod_{i \in K'_e} s^{t'} p_i \hat{s}^{t'} \right)^{\ln \left(\frac{\sum_{i \in K'_e} p_i \hat{s}^{t'}}{\sum_{i \in K'_e} p_i} \right)} \cdot \hat{s}^{t'}}{\frac{\sum_{i \in K'_e} p_i \hat{s}^{t'}}{\sum_{i \in K'_e} p_i}} \\
&= 1 - \frac{s^{t'} \cdot \hat{s}^{t'}}{\hat{s}^{t'}} = 0
\end{aligned} \tag{37}$$

Under this condition, it follows that Equation 29 equals 0, i.e.,

$$\begin{aligned}
& \sum_{i=1}^{t'} p_i s_i^{t'} \left[\sum_{j \in K'_e} p_j s_j^{t'} (v_j - v_i) - p_r s_r^{t'} (v_r - v_i) \right] \\
&= \sum_{i=1}^{t'} p_i s_i^{t'} \left[\sum_{j \in K'_e} p_j s_j^{t'} \left(1 - \frac{s_r^{t'}}{\hat{s}_r^{t'}} \cdot \frac{\hat{s}_j^{t'}}{s_j^{t'}} \right) (v_j - v_i) \right] = 0
\end{aligned} \tag{38}$$

Finally, Substituting it into Equation 28, we obtain $\Theta_{t'} = 0$. \square

Remark 10. This lemma provides a theoretical justification for prior merging strategies favoring high-similarity KV pairs (Wan et al. 2024b; Wang et al. 2024). Meanwhile, we offer an intuitive interpretation: if the merged two KV pairs are identical, i.e., $(k_e, v_e) = (k_c, v_c)$, retaining one pair and setting its Electoral Votes to 2 introduces no error in subsequent computations.

In this section, we have proven Theorem 8 and Lemma 9, which provide guarantees on the output perturbation in multi-step generation—an aspect that existing methods struggle to achieve. Moreover, our method demonstrates superior performance across various experimental evaluations. However, it should be acknowledged that predicting attention distributions further into the future is inherently challenging, leading to a significant increase in the estimated perturbation upper bound. Furthermore, the inherent input differences γ cannot be ignored, representing a fundamental problem in KV cache compression—namely, the inability to perfectly compress the KV cache into a smaller memory without any loss of information. Nevertheless, our work introduces a new perspective and analytical approach to studying KV cache eviction and merging algorithms, which we hope will inspire future research.

B Implementation Details

B.1 Models and benchmarks

Across all experiments, we utilized pre-trained model weights from Huggingface (Wolf et al. 2020). Specifically, for Llama architectures: we used the ‘huggyllama/llama-7b’ checkpoint for Llama-1-7B (Touvron et al. 2023a), ‘meta-llama/Llama-2-7b-hf’ for Llama-2-7B (Touvron et al. 2023b), and ‘meta-llama/Llama-2-13b-hf’ for Llama-2-13B. For Llama-3-8B, the ‘meta-llama/Meta-Llama-3-8B-Instruct’ checkpoint was used in the HELM evaluations, and ‘meta-llama/Meta-Llama-3-8B’ in LongBench evaluations. Regarding the Mistral architecture, we employed the ‘mistralai/Mistral-7B-Instruct-v0.3’ model (Jiang et al. 2023). For detailed information on the LongBench benchmark, please refer to its official repository (Bai et al. 2024).

B.2 Experimental Setup

Our study does not involve model training, thus no data pre-processing is required. All evaluation datasets are sourced from the publicly available lm-eval-harness (Gao et al. 2024), HELM (Narayan, Cohen, and Lapata 2018; Nallapati et al. 2016) and LongBench (Bai et al. 2024) benchmarks, and we follow their original evaluation metrics. Our implementation is primarily based on modifications to the open-source H2O and D2O codebases (Zhang et al. 2023; Wan et al. 2024b), both of which are publicly accessible. All open-source datasets, evaluation frameworks, and algorithmic implementations used in this work are employed in full compliance with their respective licenses and terms of use.

B.3 Details of Parameter Settings

For cache allocation, we adopt the default configuration provided by the open-source PyramidInfer (Yang et al. 2024a) implementation. Specifically, we follow a cosine-based decay strategy for distributing cache across layers. Within each layer, the ratio between the recent window and the heavy hitter (i.e., crucial KV entries) varies between 5:1 and 4:1. Additionally, we retain the first 4 tokens (referred to as ‘sink tokens’ in StreamingLLM (Xiao et al. 2024b)) in the cache throughout. For algorithmic hyperparameters, we adjust the merging threshold T within the range of 0.6 to 0.95, and empirically select 0.8 as the default value based on its stable performance across benchmarks. The corresponding experimental results are reported below.

Cache Budget	1%	10%
Evict-first	0.051	0.111
Merge-first	0.059	0.115

Table 3: Performance comparison with different merge policy during the prefill stage (ROUGE-2; LLAMA-7B).

Prefill’s Merge Policy The results in Table 3 show that, during the prefill stage, performing merging before applying the eviction strategy—rather than merging after eviction as is commonly done—can improve generation quality. This is because, according to Lemma 6, merging more similar items

induces smaller perturbations. In the prefill stage, there exist more tokens whose similarity exceeds the threshold; merging them first allows these tokens to be consolidated rather than evicted unnecessarily, thereby preserving more information within the limited-size cache.

Merging Thres	XSUM
0.7	0.223
0.8	0.233
0.9	0.214

Table 4: Merging threshold impact (4% compression ratio).

Merging Threshold We investigate the impact of different merging thresholds on performance. The results in Table 4 indicate that setting the merging threshold (i.e., the cosine similarity between key vectors) to 0.8 yields better performance, and its robustness has been further validated across a broader range of experiments.

C Additional Discussion

C.1 Limitations and Future Work

As analyzed in Section A, the bound on output perturbation in multi-step generation depends on two quantities: the intrinsic difference of the inputs γ , and the prediction error ϵ . The former is inherent and cannot be influenced or altered, whereas the latter is inevitably non-negligible in realistic inference scenarios. In practice, accurately predicting future attention scores is highly challenging. Prior work typically employs sliding-window averages or weighted averages as predictors, while more recent studies have begun to explore machine-learning-based approaches to obtain more accurate predictions; this appears to be a promising direction for future research.

Moreover, the merging operation inevitably incurs additional computational overhead, which can render it less favorable than purely eviction-based strategies with simpler computation in certain settings. Our mathematically derived lossless KV cache merging method, along with many other related algorithms, further relies on access to attention scores, which limits its compatibility with widely adopted inference acceleration techniques such as FlashAttention. Future work could therefore focus on designing KV cache eviction and merging algorithms that are amenable to integration with FlashAttention and related acceleration methods.

C.2 Comparison with CaM in Eliminating Output Perturbations

A closely related line of work is CaM (Zhang et al. 2024), which also analyzes the effect of KV cache eviction and its proposed merging algorithm on output perturbation, and aims to reduce or even eliminate such perturbation. However, there are two fundamental differences between CaM and our approach: (i) how “output perturbation” is defined and modeled, and (ii) how the theoretical analysis is connected to the asynchronous KV cache updates that actually occur in autoregressive decoding.

Before presenting our analysis, it is important to clarify the implementation setting shared by both CaM and our method. In practical autoregressive decoding, KV compression is always performed inside the decoding loop: the model first runs the forward pass at step t using the current KV cache and produces the output, and only afterwards applies eviction or merging operations to the cache state that will be consumed at step $t+1$ and beyond. In other words, cache compression is asynchronous with respect to the forward computation. Consequently, when we design an algorithm from the perspective of eliminating output perturbation, our first requirement is consistency at the current step: if we hypothetically recompute the step- t forward pass on the compressed KV cache, the resulting output should match that obtained from the original, uncompressed cache. Intuitively, the attention computation between the step- t query and the pre-compression KV cache should yield exactly the same result as the attention between the same query and the post-compression KV cache. This ensures that, when we “look back” at step t in the future, its behavior is indistinguishable from a world in which step t had always been computed on the compressed KV cache, without needing to refer to the original KV entries that are no longer stored. The overall decoding process thus appears temporally coherent.

In contrast, when CaM studies the effect of eviction and merging on the output at step t , it does not adopt such a “recompute on the compressed cache” viewpoint. Instead, it directly removes (or rescales) the contribution of the evicted or merged KV pair in the attention scores and in the attention-weighted output, while leaving the contributions of all remaining tokens unchanged. This effectively treats the attention scores of the other KV pairs as fixed coefficients that are not affected by modifying the cache. However, due to the softmax normalization, evicting or merging any KV pair inevitably changes the normalization term and thus the attention weights of all remaining KV pairs. Capturing this global effect of softmax is precisely why our analysis starts from the full attention expression, rather than from a model in which the other attention weights are held constant. In summary, CaM and our method formalize and analyze output perturbation from different perspectives, and each seeks to reduce or eliminate perturbation under its own definition, but our framework is explicitly aligned with the asynchronous KV-update pattern and with the impact of softmax on all tokens in the cache.