2025 21st International Conference on Mobility, Sensing and Networking (MSN) | 979-8-3315-6180-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/MSN69125.2025.00053

Efficient LLMs Inference via Similarity-Aware KV Cache Merging with Bias Calibration

Yukai Cheng†, Wenxuan Zhou†, Zhizhou Li†, Zhihao Qu†*, Baoliu Ye§*
†Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, China
§State Key Laboratory for Novel Software Technology, Nanjing University, China
{chengyukai, zhouwx, lizhizhou, quzhihao}@hhu.edu.cn, yebl@nju.edu.cn

***Abstract*—The continuously growing key-value (KV) Cache during the inference of large language models (LLMs) becomes an obstacle to efficient deployment. Recently, KV Cache compression techniques such as evicting and merging KV pairs have been widely studied, which play an important role in reducing memory usage in the decoding phase of LLM inference. However, these methods inevitably cause problems of output perturbation and computational redundancy. In this paper, we propose SimCalKV, a theoretically grounded merging strategy that identifies the optimal merging under key similarity assumptions. By averaging similar keys with a simple bias calibration and weighting values according to attention scores, SimCalKV reduces perturbation with negligible computational overhead. Experimental results across multiple models and tasks demonstrate that our approach can compress the KV Cache by up to 5× while maintaining competitive model performance, achieving up to 1.5× speedup over representative merging methods during inference. Our code is available at: https://github.com/William-kaihhu/SimCalKV.**

***Index Terms*—large language models (LLMs), KV Cache, KV Cache merging, inference speedup**

I. Introduction

Large language models (LLMs) [1]–[4] have shown remarkable performance across various domains, particularly in long-context processing and contextual reasoning. However, as the input sequence length grows and new tokens are generated during decoding, the computational cost increases significantly. To mitigate redundant computation, the key-value (KV) Cache [5]–[7] is employed to store KV pairs of previously processed tokens. The problem is that KV Cache can consume substantial memory, often exceeding the model size itself. For instance, while the OPT-30B [8] model occupies 60 GB, a batch size of 512 with a sequence length of 128 results in a KV Cache of 90 GB. Thus, reducing KV Cache memory is critical for efficient deployment.

Current mainstream approaches for KV Cache compression mainly include quantization, eviction, and merging. Quantization is one of the most direct and effective approaches, which converts high-precision numbers into lower-precision representations for storage. Related studies [9]–[12] have shown that specific quantization strategies can quantize some values down to 2-bit without obvious performance degradation, effectively reducing memory usage. Eviction and merging mechanisms have been widely studied recently. Existing works demonstrate that during both the prefill and decoding stages, some tokens are not important to the final output [13], [14]. Their KV pairs can be directly evicted or merged into other KV pairs without affecting model performance [15], [16]. Token-level merging applies the same merging method at each layer, ensuring equal numbers of key and value without affecting the computation of attention scores [16]–[19]. KVMerger [18] adopts a grouping strategy and employs a Gaussian kernel function to assign weights to the merged KV Cache. KeepKV [16] aims to achieve zero-perturbation merging through a voting mechanism. Moreover, inter-layer merging methods [20]–[23] have also been proposed to compress KV Cache with the consideration of the model architecture.

However, existing methods inevitably lead to performance degradation of LLMs. Although some efforts have been made to compensate for information loss caused by compression [16], [24], [25], errors still accumulate as inference continues and eventually degrade the performance. Excessively fine-grained compensation also leads to a substantial increase in computation time, which counteracts the throughput improvement brought by merging [16], [18]. It is crucial to balance accuracy and speedup in the efficient deployment of LLMs, so an appropriate compression strategy should be able to maintain model performance while introducing as little computational overhead as possible.

Based on the issues discussed above, we propose SimCalKV, a simple yet effective merging method that achieves KV Cache compression without introducing computational redundancy, as illustrated in Figure 1. To determine which KV pairs are suitable for merging, we first analyze token-level similarity and observe that for any key in the sequence, there typically exists another key with high cosine similarity, regardless of their positional distance. Guided by this property, we divide KV Cache into eviction and preservation sets based on attention scores, and each KV pair in the eviction set is merged with its most similar counterpart in the preservation set.

A key challenge of merging lies in mitigating the attention perturbation caused by merging. To this end, we introduce an efficient calibration mechanism. Two keys are averaged with a bias calibration, and the weights of corresponding values are derived according to their attention scores. Crucially, when two keys are highly similar, the calibration guarantees that the merged representation is theoretically equivalent to the original one, thereby ensuring that the perturbation of the attention

*Corresponding authors: Zhihao Qu and Baoliu Ye

979-8-3315-6180-2/25/$31.00 ©2025 IEEE
DOI 10.1109/MSN69125.2025.00053

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

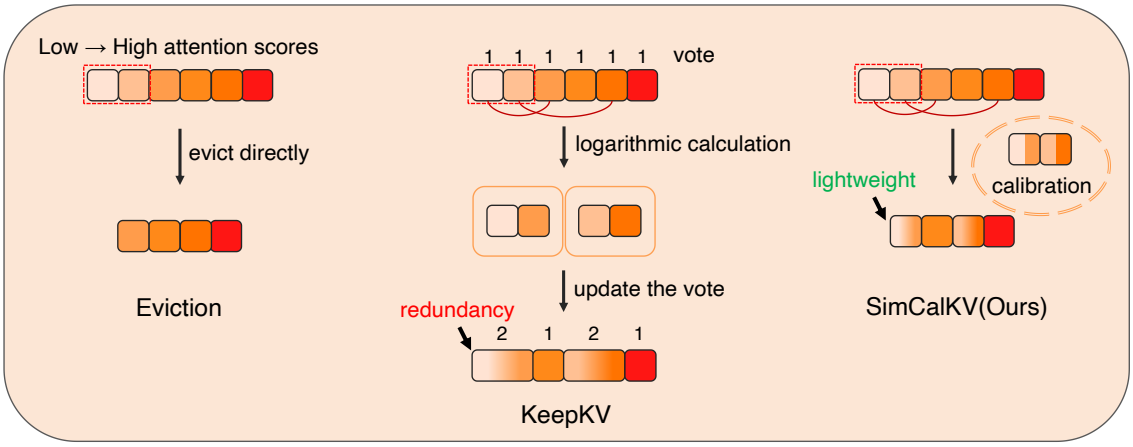


Fig. 1: Illustration of typical eviction and merging methods. Eviction directly discards certain KV pairs, and the remaining cache is used for subsequent computations. The merging strategy in KeepKV introduces a voting mechanism to achieve lossless compression, but it incurs substantial additional computation. SimCalKV mitigates output perturbation through lightweight calibration. By performing an averaging-based merge followed by a simple correction, our approach maintains model performance at a satisfactory level while achieving inference acceleration.

output is negligible.

Notably, SimCalKV introduces only negligible additional computation, and the overhead incurred is outweighed by the time saved during auto-regressive generation, resulting in a significant inference speedup. We evaluate our method on Llama and Qwen models across various tasks. Compared with state-of-the-art merging methods, our method achieves nearly the same accuracy even when compressing to low ratios such as 20% of the original. As the compression ratio increases, our inference speedup becomes more significant. Figure 2 shows that our method achieves up to 1.5× acceleration when the budget is 20% compared to KeepKV [16].

The contributions of this work are summarized as follows:

- Our theoretical analysis reveals that when two keys are similar, lossless KV Cache merging can be achieved by introducing a lightweight calibration term.
- Building upon the insight, we propose SimCalKV, a simple and efficient merging method with lightweight calibration, which applies different merging strategies to key and value, maintaining model performance and accelerating inference.
- Experiments show that our method remains competitive in accuracy compared with the state-of-the-art eviction and merging methods, while achieving significant speedup, contributing to the efficient deployment of LLMs.

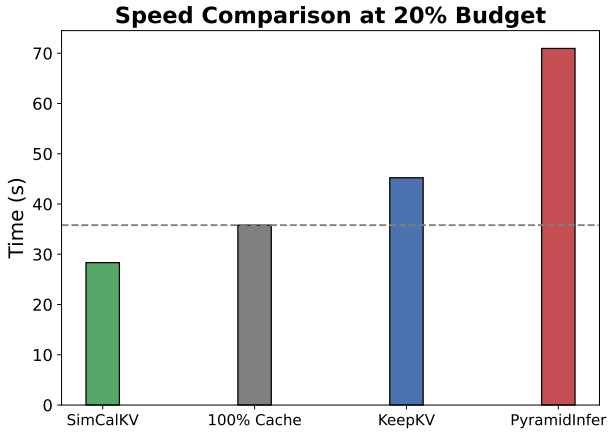


Fig. 2: The comparison of inference speed between our method and other state-of-the-art approaches shows that our method achieves significant acceleration under low budget conditions, whereas other methods introduce additional latency.

II. Related Work

A. KV Cache Compression

Existing efforts to compress the KV Cache can be broadly categorized into two directions: storage precision compression and KV pairs reduction.

a) Storage precision compression: This approach primarily relies on quantization [26]–[29], in which the high-precision values in the KV cache (typically stored in FP16 or BF16) are mapped to lower-precision formats such as INT8 or INT4 to reduce memory usage. However, naive quantization often leads to substantial performance degradation, motivating the development of more advanced techniques. For example, KIVI [10] adopts channel-wise quantization, SmoothQuant [11] introduces weight-activation rebalancing to stabilize INT8 inference, and OmniQuant [12] enhances the generalization of quantization across diverse models.

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

b) KV pairs Reduction: Eviction [30] and merging [17] strategies are the mainstream approaches to reducing the number of KV pairs. StreamingLLM [13] is a representative eviction method, which identifies an *attention sink* phenomenon: the initial tokens consistently receive high attention weights during decoding. Consequently, by retaining the KV pairs of these initial tokens while evicting those with low attention scores in the middle, StreamingLLM [13] reduces memory overhead during autoregressive generation. PyramidInfer [15] further observes that fewer KV pairs are critical for inference in deeper layers, and thus progressively decreases the retention ratio with depth. For merging strategies, most approaches build on top of eviction by merging the KV pairs that would otherwise be discarded into the remaining ones, thereby mitigating information loss. Intra-layer token-wise methods are common, such as CaM [17] and KVMerger [18]. There are also inter-layer approaches. MiniCache [21] observes the similarity of the same token's KV pairs across adjacent layers and applies spherical linear interpolation for merging. xKV [23] demonstrates cross-layer subspace similarity via CKA [31] analysis, and employs cross-layer SVD decomposition to achieve high compression ratios with minimal performance loss.

B. Model Calibration

As mentioned above, KV Cache compression inevitably incurs performance degradation to some extent. Therefore, some works explore methods for repairing or calibrating the model outputs after compression. For example, The Unreasonable Ineffectiveness of the Deeper Layers [24] mitigates the loss from layer pruning by applying QLoRA fine-tuning. CLA [25] retrains the model after reusing adjacent-layer KV caches to maintain performance, and KeepKV [16] proposes a voting mechanism that enables "zero-perturbation" merging and replaces attention score with EMA score [32], [33] to achieve multi-step generation.

However, these approaches are either not training-free or require additional computation. The overall inference time increases as the compression ratio grows. To address these issues, we propose SimCalKV to calibrate output perturbations through simple matrix addition operations, which accelerates LLM inference while maintaining model performance under low KV cache budgets.

III. Methodology

A. Preliminary

1) Attention mechanism: Attention mechanism is the core part of Transformer models [34], and the KV cache plays a crucial role in it. Each token in an LLM is encoded as a vector $\mathbf{x} \in \mathbb{R}^d$. By multiplying $\mathbf{x}$ with the weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$, the corresponding query, key, and value vectors are obtained. These matrices represent the semantic information of the token. When predicting the next token, the LLM depends on all previous tokens. Since computing the keys and values for previous tokens at each time step is redundant, we can store these $\mathbf{K}$ and $\mathbf{V}$ vectors as KV cache which avoids repeated computation and significantly improves inference efficiency.

In this section, we introduce the basic workflow of how KV cache participates in the computation. For simplicity, we consider the attention mechanism of a single head in one layer of the LLM. Suppose at time step t, the input of the L-th layer is denoted as $\mathbf{X}_t = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t]$. The attention weight matrices are $\mathbf{W}_q \in \mathbb{R}^{d\times d}$, $\mathbf{W}_k \in \mathbb{R}^{d\times d}$, $\mathbf{W}_v \in \mathbb{R}^{d\times d}$, where d is the hidden dimension. Then we have:

$$\begin{aligned} \mathbf{K}_t &= \mathbf{X}_t\mathbf{W}_k = [\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_t], \\ \mathbf{V}_t &= \mathbf{X}_t\mathbf{W}_v = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t], \\ \mathbf{q}_t &= \mathbf{x}_t\mathbf{W}_q. \end{aligned} \tag{1}$$

When the KV cache is used, the keys and values computed from the previous $t-1$ time steps are stored. At time step t, the newly generated $\mathbf{k}_t$ and $\mathbf{v}_t$ are concatenated to the KV cache to form $\mathbf{K}_t$ and $\mathbf{V}_t$. Then, the query $\mathbf{q}_t$ at the current step is used to compute attention scores with the entire KV matrices, yielding the attention output. The subsequent decoding process repeats the above procedure, where each newly generated KV vector is appended to the existing KV matrices, and the cache is continuously accumulated and updated.

$$\begin{aligned} \mathbf{K}_{t-1} &= [\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_{t-1}], \\ \mathbf{V}_{t-1} &= [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{t-1}], \\ \mathbf{K}_t &= \mathrm{Concat}(\mathbf{K}_{t-1}, \mathbf{k}_t), \\ \mathbf{V}_t &= \mathrm{Concat}(\mathbf{V}_{t-1}, \mathbf{v}_t). \end{aligned} \tag{2}$$

The attention weights of the *i*-th token can be written as:

$$w_i = e^{\frac{\mathbf{q}_t\mathbf{k}_i^\top}{\sqrt{d}}}, \quad A_i = \mathrm{softmax}\left(\frac{\mathbf{q}_t\mathbf{k}_i^\top}{\sqrt{d}}\right) = \frac{w_i}{\sum_j w_j}. \tag{3}$$

and the attention output is

$$\mathbf{O}_t = \sum_{i=1}^{t} A_i\mathbf{v}_i. \tag{4}$$

The KV Cache effectively alleviates redundant computations. However, the memory overhead caused by the KV cache grows rapidly as the time step increases. The matrix multiplication operations also become increasingly expensive and the throughput declines with the sequence length, making efficient deployment challenging. Therefore, it is crucial to design a proper KV compression strategy to reduce memory usage while maintaining competitive model performance.

2) KV Cache Merging: The merging of KV cache mainly consists of two steps. Take the key matrix $\mathbf{K}$ as an example. Let $\mathcal{K} = \{\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_t\}$ denote the set of key vectors, where each $\mathbf{k}_i$ corresponds to a row of $\mathbf{K}$.

(*i*) Split $\mathcal{K}$ into two disjoint subsets $\mathcal{K}_m$ and $\mathcal{K}_n$ according to the attention scores. Here, $\mathcal{K}_m$ denotes the important tokens that should be preserved, while the tokens in $\mathcal{K}_n$ need to be merged into $\mathcal{K}_m$, such that $\mathcal{K} = \mathcal{K}_m \cup \mathcal{K}_n$ and $\mathcal{K}_m \cap \mathcal{K}_n = \emptyset$.

(*ii*) For each $\mathbf{k}_i \in \mathcal{K}_n$, apply a merging strategy to update some $\mathbf{k}_j \in \mathcal{K}_m$. We denote this as $\mathbf{k}_r = \mathrm{update}(\mathbf{k}_j)$, where the update function can be viewed as a merging function $\mathrm{Merge}(\mathbf{k}_i, \mathbf{k}_j)$. This process continues until all tokens in $\mathcal{K}_n$ are merged into $\mathcal{K}_m$.

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

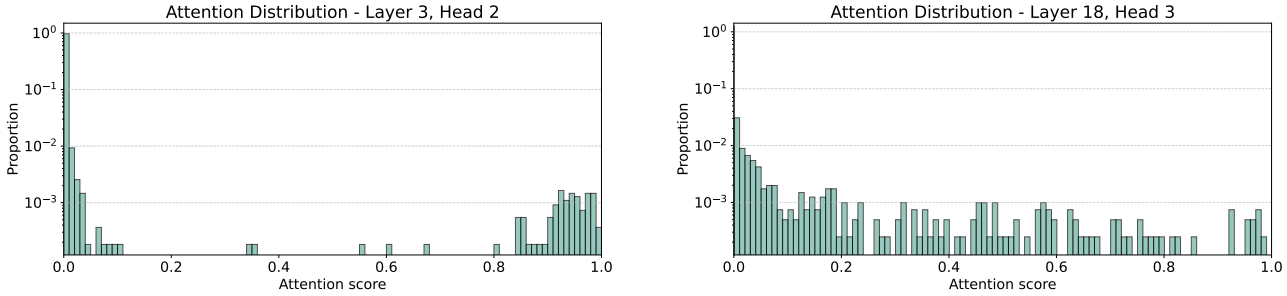


Fig. 3: Attention distribution of the LLama2-7B model on the OpenBookQA dataset. By randomly selecting different layers and attention heads, we found that over 90% of the tokens receive very low attention scores, mostly in the range of 0 to 0.01. Tokens with large attention scores account for only a small portion.

The merging function is defined as:

$$\text{Merge}(\mathbf{k}_i, \mathbf{k}_j) = w_i \mathbf{k}_i + w_j \mathbf{k}_j, \tag{5}$$

where the weights w_i and w_j play an important role in determining the quality of subsequent outputs.

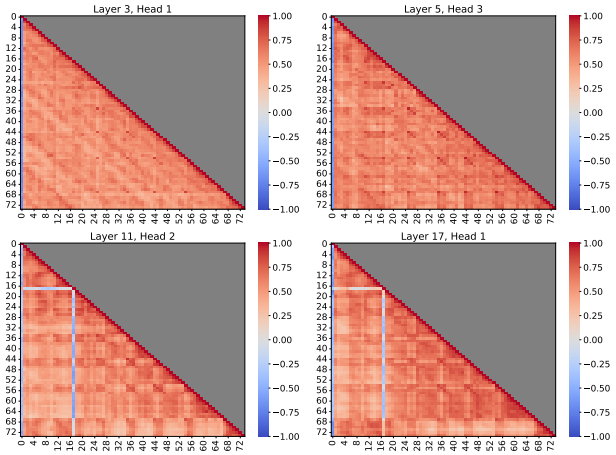
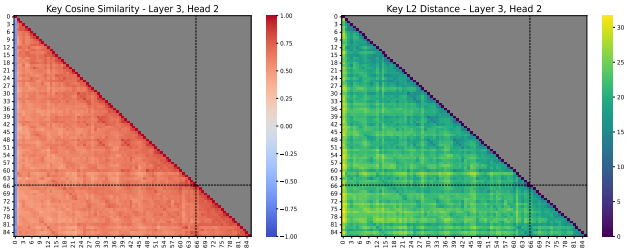


Fig. 4: Token-wise key similarity across different layers and heads of LLama2-7B. The horizontal and vertical axes represent token indices. The result shows that a substantial number of key pairs exhibit high similarity.

B. Motivation

Before executing the merging algorithm, we first need to determine the eviction set $\mathcal{K}_n$ and the preservation set $\mathcal{K}_m$. Previous research has revealed the sparsity of attention distributions, where only a small fraction of tokens receive high scores, while most have scores close to zero. Inspired by StreamingLLM [13], we conduct relative experiments, and the results are shown in Figure 3. Tokens with low attention scores account for over 90% of the sequence. They play a relatively unimportant role and can therefore be merged into tokens with high attention scores. Furthermore, for each $\mathbf{k}_i \in \mathcal{K}_n$, we need to identify the most appropriate $\mathbf{k}_j \in \mathcal{K}_m$ to merge with.

1) Key Pairs Similarity: Suppose there are two identical keys $\mathbf{k}_1$ and $\mathbf{k}_2$. They will receive exactly the same attention score from the same $\mathbf{q}$, i.e., $A_1 = A_2$. In this case, we can attempt to use a single key to replace both of them, thereby reducing memory consumption while minimally affecting the attention distribution. Consequently, we propose the following assumption: **The correction of attention outputs before and after merging requires that the two key vectors to be merged, $\mathbf{k}_m$ and $\mathbf{k}_n$, are similar.**



(a) Cosine similarity (b) L_2 distance

Fig. 5: Correlation between cosine similarity and L_2 distance of key vectors. The tokens on the left and right sides of the dashed line correspond to the prefill and decoding stages, respectively.

To measure the similarity between $\mathbf{k}_1$ and $\mathbf{k}_2$, we need to compute two metrics: cosine similarity and Euclidean (L_2) distance. The cosine similarity and L_2 distance between $\mathbf{k_i}$ and $\mathbf{k_j}$ are defined as follows:

$$\text{cosSimilarity}(\mathbf{k}_i, \mathbf{k}_j) = \frac{\mathbf{k}_i \mathbf{k}_j^\top}{\|\mathbf{k}_i\| \, \|\mathbf{k}_j\|}, \tag{6}$$

$$D_{L_2}(\mathbf{k}_i, \mathbf{k}_j) = \|\mathbf{k}_i - \mathbf{k}_j\|_2. \tag{7}$$

We conduct a comprehensive experimental analysis based on both cosine similarity and L_2 distance across all layers and heads of the LLama model and visualize part of the results in Figure 4. The findings show that keys exhibit a high degree of similarity in the majority of layers and heads. Moreover, such similarity is not limited to adjacent keys. Two keys that are far apart can also have very high cosine similarity. As

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

shown in Figure 5, the distributions of cosine similarity and L_2 distance exhibit a strong correlation. Pairs of key vectors with high cosine similarity tend to have small L_2 distances. In other words, there are no cases where two vectors have a small angle but a large Euclidean distance. Consequently, we can use cosine similarity alone to measure the correlation between two keys.

2) Output Perturbation: The direct reason for the degraded model performance after merging is the change in the attention distribution. This change accumulates over time and eventually becomes large enough to affect the model's output. Suppose we merge $\mathbf{k}_m, \mathbf{k}_n$ into $\mathbf{k}_r$, and $\mathbf{v}_m, \mathbf{v}_n$ into $\mathbf{v}_r$. Let S denote the sum of all other weights except the tokens being merged.

$$S = \sum_{i=1}^{t} w_i - (w_m + w_n). \tag{8}$$

After merging, the perturbation to the attention output can be expressed as

$$P = \frac{A_r \mathbf{v}_r}{S + w_r} - \frac{A_m \mathbf{v}_m + A_n \mathbf{v}_n}{S + w_m + w_n}. \tag{9}$$

To mitigate the effect of merging, we aim to select an optimal merging rule f^* from the candidate set $\mathcal{F}$ such that the perturbation is minimized:

$$f^* = \arg\min_{f \in \mathcal{F}} |P(f)|. \tag{10}$$

Here $\mathcal{F}$ denotes the set of possible merging strategies, and $P(f)$ is the resulting perturbation under strategy f. Any merging algorithm inevitably introduces additional computational overhead. It mainly includes the attention calculation, similarity measurement and the merging operation itself. If the overhead is too large, it may offset the speedup gained from cache reduction, causing the total inference time to increase instead of decrease. The major challenge lies in that: simple merging formulas are likely to degrade accuracy, while complex ones introduce significant latency. Therefore, it is crucial to find a balance: **how can we maintain nearly the same level of accuracy while improving inference speed?**

C. Method: SimCalKV

Based on the above considerations, we propose a simple yet efficient merging method that accelerates inference while preserving model performance.

1) Set Partitioning: At time step t, we compute the cumulative attention each token receives from all other tokens and rank them in descending order. Given a compression ratio R, the eviction set $\mathcal{K}_n$ is formed by selecting $\lfloor t \cdot R \rfloor$ key vectors with the lowest attention, while the preservation set $\mathcal{K}_m$ consists of the remaining $\left(t - \lfloor t \cdot R \rfloor\right)$ key vectors.

2) Merging: To solve Equation (10), we need to design a merging method with low computational overhead. Suppose we need to merge $\mathbf{k}_m$, $\mathbf{k}_n$ into $\mathbf{k}_r$, and $\mathbf{v}_m$, $\mathbf{v}_n$ into $\mathbf{v}_r$, where $\mathbf{k}_m$ and $\mathbf{v}_m$ come from the preservation set and $\mathbf{k}_n$ and $\mathbf{v}_n$ come from the eviction set. Let the attention output before and after merging be denoted as $\mathbf{O}$ and $\mathbf{O}'$, respectively. The original attention output can be expressed as

$$\mathbf{O} = \frac{\sum_{i=1, i\neq m,n}^{t} w_i \mathbf{v}_i + w_m \mathbf{v}_m + w_n \mathbf{v}_n}{\sum_{i=1, i\neq m,n}^{t} w_i + w_m + w_n}, \tag{11}$$

and the output after merging is given by

$$\mathbf{O}' = \frac{\sum_{i=1, i\neq m,n}^{t} w_i \mathbf{v}_i + w_r \mathbf{v}_r}{\sum_{i=1, i\neq m,n}^{t} w_i + w_r}. \tag{12}$$

It should be noted that r is not a subscript. The merged key-value pair $(\mathbf{k}_r, \mathbf{v}_r)$ will be inserted at the same position as $(\mathbf{k}_m, \mathbf{v}_m)$, since $\mathbf{k}_m$ and $\mathbf{v}_m$ come from the preservation set. The ideal case requires that $\mathbf{O} = \mathbf{O}'$. Thus, we can derive the following equation:

$$e^{\frac{\mathbf{q}\mathbf{k}_m^\top}{\sqrt{d}}} + e^{\frac{\mathbf{q}\mathbf{k}_n^\top}{\sqrt{d}}} = e^{\frac{\mathbf{q}\mathbf{k}_r^\top}{\sqrt{d}}}, \quad w_m \mathbf{v}_m + w_n \mathbf{v}_n = w_r \mathbf{v}_r. \tag{13}$$

Now consider the special case when the two keys are identical, i.e., $\mathbf{k}_m = \mathbf{k}_n$. In this case, Equation (13) reduces to

$$e^{\frac{\mathbf{q}\mathbf{k}_r^\top}{\sqrt{d}}} = 2e^{\frac{\mathbf{q}\mathbf{k}_m^\top}{\sqrt{d}}}. \tag{14}$$

The approximation error $|\mathbf{O} - \mathbf{O}'|$ mainly depends on the similarity between $\mathbf{k}_m$ and $\mathbf{k}_n$. The more similar the two keys are, the smaller the error will be. Therefore, for each candidate key $\mathbf{k}_n$, we always merge it with the key $\mathbf{k}_m$ that has the highest similarity. This strategy ensures that the merging process introduces minimal error. Based on the observation, we propose separate merging formulas as follows for both keys and values, which are simple and introduce very little additional computational overhead.

$$\mathbf{k}_r = \frac{\mathbf{k}_m + \mathbf{k}_n}{2} + \boldsymbol{\delta}, \quad \mathbf{v}_r = \frac{w_m \mathbf{v}_m + w_n \mathbf{v}_n}{w_m + w_n}. \tag{15}$$

Here, $\boldsymbol{\delta}$ is introduced as a bias vector with the same shape as $\mathbf{k}_r$, where each entry is set to $\ln 2$.

$$\boldsymbol{\delta} = [\ln 2, \ln 2, \ldots, \ln 2] \in \mathbb{R}^d. \tag{16}$$

3) The derivation of bias vector δ: According to Equation (13), we need to solve the equation

$$r = \ln(e^m + e^n). \tag{17}$$

To facilitate the subsequent calculations, we introduce two new variables x and y.

$$x = \frac{m+n}{2}, \quad y = \frac{m-n}{2},$$

Using the new variables, we derive the following closed-form expression for r. When the difference between m and n is sufficiently small, a Taylor expansion of $\ln(\cosh y)$ can be applied to approximately express r.

$$\begin{aligned} r &= x + \ln 2 + \ln(\cosh y) \\ &\approx \frac{m+n}{2} + \ln 2 + \frac{(m-n)^2}{8} + O(y^4) \end{aligned} \tag{18}$$

This shows that the merged exponent r can be decomposed into two parts: a linear mean term x and a nonlinear correction

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

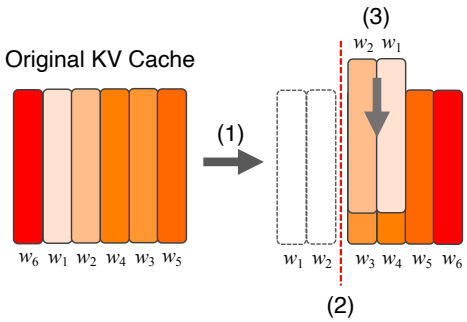


Fig. 6: Illustration of SimCalKV merging. (1) Sort according to attention scores. (2) Partition into retained set and evicted set. (3) Merge and calibration. KV Cache is updated while maintaining the original order.

term $\ln(\cosh y)$ that depends on the difference between m and n. The above analysis and Equation (18) motivate the merging formula for $\mathbf{k}_r$. According to Equation (14), to ensure

$$\frac{\mathbf{q}(\mathbf{k}_m^\top + \boldsymbol{\delta})}{\sqrt{d}} = \ln 2 + \frac{\mathbf{q}\mathbf{k}_m^\top}{\sqrt{d}}, \tag{19}$$

it is required that the sum of all elements in $\mathbf{q}$, denoted as $\sum_i q_i$, satisfies the following condition

$$\delta_k \cdot \frac{\sum_i q_i}{\sqrt{d}} = \ln 2. \tag{20}$$

Notation δ_k denotes the value of each element in $\boldsymbol{\delta}$. However, since the query vector $\mathbf{q}$ varies at each decoding step, the sum of its elements also changes accordingly. As a result, δ_k obtained from Equation (20) can only satisfy the merging condition for the current time step. If δ_k is set as a learnable parameter, it would introduce additional computational overhead with limited performance enhancement. Moreover, our experimental results further demonstrate that using $\ln 2$ yields higher accuracy compared with δ_k solved by Equation (20). Based on this output, we select appropriate key-value pairs for compression. The correction after the averaged merging remains valid at any time step. The merging process is illustrated in Figure 6 and the algorithm is described in Algorithm 1.

Algorithm 1 SimCalKV Merging at t-th Step

Input: Attention scores w, compression ratio R, KV cache
Compute cumulative attention for each key vector:
$A_k = \sum_{\mathbf{q}} \mathrm{softmax}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d}}\right),\ \forall \mathbf{k} \in \mathcal{K}$
Partition $\mathcal{K}$ into preserved set $\mathcal{K}_m$ and eviction set $\mathcal{K}_n$ according to $\{A_k\}$ and R
for each $\mathbf{k}_i \in \mathcal{K}_n$ **do**
    Compute similarity with preserved keys:
    $C = \mathrm{cosSimilarity}(\mathbf{k}_i, \mathbf{k}_j),\ \forall \mathbf{k}_j \in \mathcal{K}_m$
    Select the most similar preserved key:
    $\mathbf{k}_j = \arg\max_{\mathbf{k}_j \in \mathcal{K}_m} C$
    Merge:
    $\mathbf{k}_r = \frac{\mathbf{k}_j + \mathbf{k}_i}{2} + \boldsymbol{\delta}, \quad \mathbf{v}_r = \frac{w_j \mathbf{v}_j + w_i \mathbf{v}_i}{w_j + w_i}$
    Evict $(\mathbf{k}_i, \mathbf{v}_i)$ and replace $(\mathbf{k}_j, \mathbf{v}_j)$ with $(\mathbf{k}_r, \mathbf{v}_r)$ in KV cache
end for
Output: Updated KV cache

IV. Experiment

A. Experimental Settings

a) Tasks: We evaluate SimCalKV on a comprehensive set of tasks, covering both standard and extended context lengths. Specifically, we choose the COPA [35] and OpenBookQA [36] datasets for question-answering. For summarization, we employ the XSUM [37] and CNN/DailyMail [38] tasks provided by the HELM framework. To assess performance on long-context and synthetic tasks, we utilize LongBench [39] and Needle-in-the-Haystack [40], which examine the algorithm's ability to handle extended sequences, multi-document reasoning, and retrieval-intensive scenarios.

b) Models and Baselines: Our evaluation is conducted on four large language models of different scales: OPT-1.3B [8], TinyLlama [41], Qwen2.5-7B-Instruct [42], Llama2-7B [2] and LLama3.1-8B [43]. We compare our method against multiple baseline approaches for KV cache management, including representative eviction and merging methods **PyramidInfer** [15], **CaM** [17] and **KeepKV** [16].

c) Implementation: In our experiments, we apply the same compression ratio and strategy to each layer of the LLM. All models are implemented using the Hugging Face Transformers [44] library and evaluated on NVIDIA RTX 4090 24GB and H20 SXM 96GB GPUs. Performance metrics contain total inference time, throughput, memory consumption and task accuracy, providing a comprehensive assessment of the algorithm's effectiveness across different models and datasets.

Budget	Full Cache	KeepKV	PyramidInfer	SimCalKV(Ours)
80%	28.88	29.18	29.45	**29.68**
50%	24.19	26.05	25.77	**27.17**
20%	20.35	26.90	26.41	**28.03**

TABLE I: Throughput (tokens/s) comparison under different compression ratios. We select different models and datasets for each compression ratio, so the throughput of the full cache also varies. From the experimental results, it can be observed that the throughput exhibits a positive correlation with the compression ratio.

B. Inference Time and Throughput Test

Inference time refers to the total time consumed, including prefill, decoding and merging. Most previous studies have focused primarily on throughput. When compression is applied, the reduction in matrix multiplication operations naturally improves throughput as shown in Table I. But the total runtime of the model remains an important evaluation metric. Therefore, we measure inference time under various conditions, including KV Cache budgets ranging from 100%

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

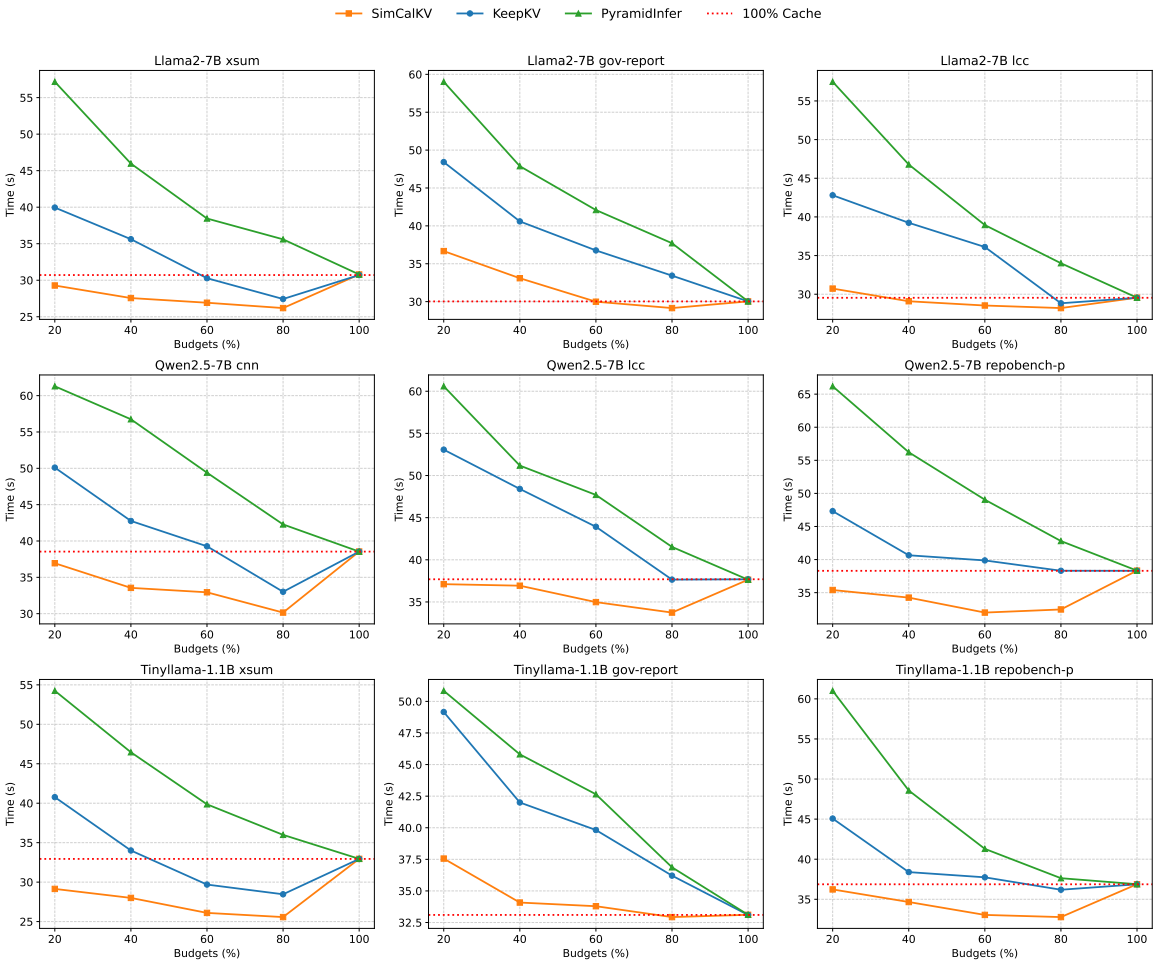


Fig. 7: Comparison of inference time using different models and datasets. The figure shows that compared with full-cache inference, our compression method achieves practically meaningful acceleration on various task. The speedup can reach up to **1.5×**, which is particularly valuable for long-context tasks.

to 20% and different sequence lengths. Since the merging algorithm itself inevitably introduces extra computation, the total inference time may even increase when the sequence is short and the compression ratio is high. However, our method reduces the computational overhead, allowing it to complete inference faster under all circumstances. When the budgets are smaller, the acceleration is more obvious. The total inference time increases with compression ratio because more merging computation is required. SimCalKV has the capability to slow down the rate of time growth as shown in Figure 7.

To better illustrate the efficiency of our algorithm in compressing the KV Cache, we report the time overhead of the compression process alone, excluding the decoding stage. We use the gov_report dataset from LongBench as a representative example due to its relatively long input sequences. The input and output lengths are set to 4096 and 512, and three compression ratios of 0.2, 0.5, and 0.8 are evaluated. The results are presented in Table II. A larger compression ratio requires more merging operations, which increases the processing time.

Compression Ratio	Evaluation	Methods		
		SimCalKV	KeepKV	CaM
0.2×	time	**4.50s**	4.74s	5.31s
	score	17.21	**18.56**	18.01
0.5×	time	**4.84s**	6.26s	6.13s
	score	**18.81**	15.99	17.39
0.8×	time	**5.11s**	5.50s	7.26s
	score	12.98	10.41	**17.41**

TABLE II: Comparison of different compression methods on H20 SXM 96GB. The compression time of SimCalKV is the shortest and the Rouge-L score remains competitive.

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

Model	Budget	Method	xsum	cnn	gov-report	lcc	repobench-p	copa	openbookqa	avg
Qwen2.5-7B	100%	Full Cache	6.50	8.32	19.28	43.43	33.75	75.43	32.59	31.33
	60%	PyramidInfer	5.13	6.57	18.84	41.20	31.09	64.92	24.17	27.41
		KeepKV	6.54	7.47	**19.15**	**42.90**	32.03	75.11	31.34	**30.65**
		CaM	**6.58**	7.31	18.94	42.40	32.11	**75.82**	31.00	30.59
		SimCalKV	6.38	**7.53**	19.01	42.47	**32.62**	74.93	**31.45**	30.63
	20%	PyramidInfer	4.22	5.09	18.21	40.05	29.39	40.08	12.49	21.36
		KeepKV	**6.35**	7.47	**18.94**	**41.96**	32.00	73.83	30.36	**30.13**
		CaM	6.12	**7.58**	18.62	40.85	27.75	73.51	**30.91**	29.33
		SimCalKV	6.19	7.26	18.86	39.12	**32.76**	**73.99**	30.21	29.77
LLaMA2-7B	100%	Full Cache	8.86	13.30	26.34	60.41	49.79	87.00	31.40	39.59
	60%	PyramidInfer	8.42	12.78	25.59	57.09	48.46	83.01	29.85	37.89
		KeepKV	**8.91**	13.12	25.68	**60.33**	49.60	86.58	**30.60**	39.26
		CaM	8.60	12.95	25.74	59.81	49.30	87.90	30.22	39.22
		SimCalKV	8.30	**13.18**	**26.34**	60.26	**49.65**	**86.77**	30.53	**39.29**
	20%	PyramidInfer	8.17	12.52	**25.32**	52.49	48.10	79.36	26.68	36.09
		KeepKV	8.05	**12.93**	25.20	57.42	**48.84**	82.44	28.75	37.66
		CaM	8.24	10.63	24.85	56.80	48.33	81.92	28.20	37.00
		SimCalKV	**8.51**	12.77	24.41	**58.22**	48.21	**83.06**	**28.80**	**37.71**
LLaMA3.1-8B	100%	Full Cache	14.48	29.74	26.52	63.04	63.39	50.00	57.80	43.57
	60%	PyramidInfer	11.86	23.13	28.54	47.31	62.14	38.00	47.45	36.92
		KeepKV	13.55	26.35	**28.69**	48.94	61.98	**47.80**	**50.58**	**39.70**
		CaM	13.38	**29.03**	28.43	44.49	**62.40**	34.30	46.43	36.92
		SimCalKV	**13.70**	25.86	28.08	**50.51**	62.17	46.90	43.70	38.70
	20%	PyramidInfer	12.36	16.91	17.97	38.30	62.93	25.00	**45.70**	31.31
		KeepKV	9.32	15.21	20.28	38.17	**62.93**	20.10	40.90	29.56
		CaM	**13.48**	**20.69**	18.93	36.47	61.28	35.20	33.88	31.41
		SimCalKV	10.56	12.13	**22.31**	**39.85**	62.91	**36.60**	35.77	**31.44**

TABLE III: Performance comparison of KV compression methods on different models under various budgets. Our method remains competitive under low KV Cache budgets.

Dataset	Full Cache	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5	0
xsum	11.69	10.78	10.21	9.91	10.66	10.77	10.57	10.64	10.78	10.92	10.34	10.66
cnn_dailymail	20.70	13.76	13.61	12.69	12.58	12.64	12.52	13.83	13.17	13.15	12.85	12.90

TABLE IV: Rouge-L scores of LLama3.1-8B on different tasks under various similarity thresholds, with a compression ratio of 0.4×. The input and output lengths are 512+64 for XSUM and 1024+64 for CNN/DailyMail.

C. Accuracy under Various Compression Ratios

We evaluate the performance of SimCalKV on the LLama and Qwen model series, using KeepKV and PyramidInfer as baselines for comparison. The results are shown in Table III. Across different compression ratios, our method is able to maintain competitive accuracy on various tasks. Eviction-based approaches suffer from significant degradation under lower budgets, while the state-of-the-art merging method can largely mitigate the performance drop, especially at very low budgets. Since our approach relies on similarity, there may not be enough similar key pairs to merge when the compression ratio becomes too large, which inevitably causes some accuracy loss. Nevertheless, the experimental results show that this degradation is only slightly weaker than KeepKV. In short, we trade a small portion of accuracy for a substantial speedup, as will be demonstrated in the experiments presented in the next section. The trade-off highlights the practicality of our method in real-world deployment scenarios.

D. Similarity Thresholds of SimCalKV

Previous research [13], [15] have shown that directly evicting tokens with low attention scores can maintain a certain level of accuracy. However, to further improve model performance, subsequent research adopts similarity-based merging. The question then arises: **what degree of similarity between keys is appropriate for merging?** To address this issue, we conducted relative experiments to compare the accuracy under

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

different similarity thresholds. Specifically, the keys of two tokens are merged when their similarity exceeds the threshold. Otherwise, we evict the token directly. We vary the similarity threshold from 0.95 to 0.5 with an interval of 0.05. Experimental results in Table IV indicate that the model performance does not exhibit a clear dependency on the specific threshold value. For instance, on the XSUM dataset, the Rouge-L score fluctuates slightly as the threshold changes. Even when the threshold is set to 0 (i.e., all keys are merged), the model demonstrates stable performance. This observation suggests that it is sufficient to adhere to the principle of merging each key in the eviction set with its most similar counterpart in the retention set, without being overly concerned about the precise magnitude of their similarity.

Compression Ratio	Value of δ	rouge1	rouge2	rougeL
Full Cache	-	16.73	3.00	11.72
0.4×	$\boldsymbol{\delta_k}$**(2.05)**	12.08	1.88	9.71
	ln2	**14.41**	**2.16**	**10.68**
0.8×	$\boldsymbol{\delta_k}$**(2.05)**	10.18	1.01	8.37
	ln2	**13.62**	**2.78**	**10.93**

TABLE V: Rouge scores of LLaMA3.1-8B on XSUM under various bias vectors. The value 2.05 is computed accurately according to Equation (21). When each element of the bias vector δ is set to ln2, the model performance under different compression ratios is the closest to that of the full cache.

E. Ablation study on the bias vector δ

To accurately solve for δ_k from Equation (20), we need to know all $\mathbf{q}$ vectors in the decoding steps in advance. Assume the generated sequence length during decoding is l. Timestep t is the moment when the model starts decoding. Then, we solve the following equation based on $\mathbf{q}_t$ to $\mathbf{q}_{t+l}$:

$$\delta_k \cdot \frac{\frac{1}{l}\sum_{i=t}^{t+l}\sum_j q_{i,j}}{\sqrt{d}} = \ln 2, \tag{21}$$

where $\sum_j q_{i,j}$ denotes the sum of all elements in the i-th query vector. Different datasets, models, and decoding lengths may lead to different values of δ_k, indicating that the algorithm lacks universality. We take the XSUM dataset as an example to verify whether the δ_k obtained by the above method can improve model accuracy. The results are shown in Table V. Since query vectors in the future are unknown during the decoding process. At the current timestep t, only $\mathbf{q}_t$ is available, making Equation (21) unsolvable in practical deployment.

F. Analysis

In our experiments on inference latency, the results show that the strategy of adopting different compression ratios across layers, as used in PyramidInfer [15], leads to a significant increase in inference time. This is likely because varying the size of the KV cache at each layer requires dynamic memory allocation and reduces GPU kernel reusability. Therefore, we applied the same compression ratio for KeepKV [16] and SimCalKV across all layers in our final experimental setup, which is more hardware-friendly. Our method maintains competitive accuracy while still achieving over 1.3× acceleration when more than 20% of the KV Cache is preserved.

Another interesting phenomenon is that the model performance does not strictly degrade as the amount of retained KV Cache decreases. In theory, each token's KV Cache stores important information. Compressing too much of the KV Cache could prevent the model from properly understanding the semantics, thereby reducing task accuracy. However, experiments show that selecting a relatively large compression ratio still allows the model to maintain competitive performance. We use an interval of 0.1 and evaluate LLama3.1-8B under KV Cache budgets ranging from 0.2 to 0.9 on the cnn_dailymail dataset. The results are shown in Figure 8. These findings inspire us to explore more suitable compression strategies and budget ratios in the future to maintain model performance.

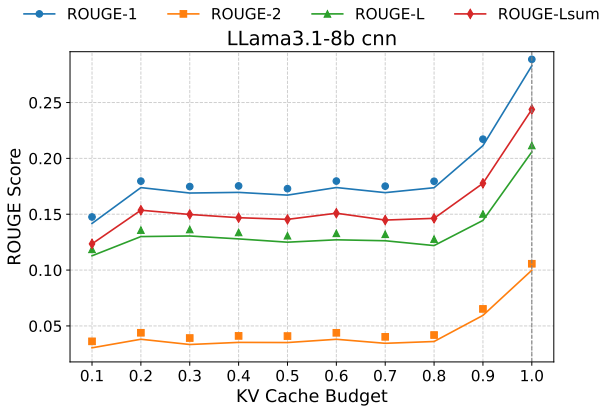


Fig. 8: Model performance remains relatively stable for KV Cache budgets in the range of 0.2 to 0.8. Even under lower budgets, the model can maintain reasonably good performance.

V. Conclusion

In this paper, we propose a simple KV Cache merging method. The key is calibrated by adding a bias vector after averaging, while the value is merged according to the attention scores, which reduces output perturbation and allows the model to generate nearly lossless outputs. At the same time, since our method does not introduce significant additional computational overhead, it achieves substantial reductions in total inference time across different input lengths and compression ratios compared with the latest approaches, resulting in considerable speedup.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan *et al.*, "Language models are few-shot learners," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[3] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[4] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds and machines*, vol. 30, no. 4, pp. 681–694, 2020.

[5] L. Haoyang, Y. Li, A. Tian, T. Tang, Z. Xu, X. Chen *et al.*, "A survey on large language model acceleration based on KV cache management," *Transactions on Machine Learning Research*, 2025.

[6] S. Luohe, H. Zhang, Y. Yao, Z. Li, and H. Zhao, "Keep the cost down: A review on methods to optimize LLM's KV-cache consumption," in *Proceedings of the Conference on Language Modeling*, 2024.

[7] N. Javidnia, B. D. Rouhani, and F. Koushanfar, "Key, value, compress: A systematic exploration of KV cache compression techniques," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 2025.

[8] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen *et al.*, "OPT: Open pre-trained transformer language models," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[9] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer *et al.*, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024, pp. 1270–1303.

[10] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman *et al.*, "KIVI: A tuning-free asymmetric 2bit quantization for kv cache," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[11] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2023, pp. 38 087–38 099.

[12] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li *et al.*, "Omniquant: Omnidirectionally calibrated quantization for large language models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[13] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[14] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai *et al.*, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 34 661–34 710.

[15] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao, "Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 3258–3270.

[16] Y. Tian, Z. Wang, Y. Peng, A. Yuan, Z. Wang, B. Yi *et al.*, "KeepKV: Eliminating output perturbation in kv cache compression for efficient llms inference," *arXiv preprint arXiv:2504.09936*, 2025.

[17] Y. Zhang, Y. Du, G. Luo, Y. Zhong, Z. Zhang, S. Liu, and R. Ji, "Cam: Cache merging for memory-efficient llms inference," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[18] Z. Wang, B. Jin, Z. Yu, and M. Zhang, "Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks," *arXiv preprint arXiv:2407.08454*, 2024.

[19] Z. Wan, X. Wu, Y. Zhang, Y. Xin, C. Tao, Z. Zhu *et al.*, "D2o: Dynamic discriminative operations for efficient long-context inference of large language models," 2025.

[20] H. Wu and K. Tu, "Layer-condensed kv cache for efficient inference of large language models," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024, pp. 11 175–11 188.

[21] A. Liu, J. Liu, Z. Pan, Y. He, G. Haffari, and B. Zhuang, "Minicache: Kv cache compression in depth dimension for large language models," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024, pp. 139 997–140 031.

[22] C.-C. Chang, W.-C. Lin, C.-Y. Lin, C.-Y. Chen, Y.-F. Hu, P.-S. Wang *et al.*, "Palu: Kv-cache compression with low-rank projection," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[23] C.-C. Chang, C.-Y. Lin, Y. Akhauri, W.-C. Lin, K.-C. Wu, L. Ceze *et al.*, "xKV: Cross-layer svd for kv-cache compression," *arXiv preprint arXiv:2503.18893*, 2025.

[24] A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. Roberts, "The unreasonable ineffectiveness of the deeper layers," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[25] W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. Ragan-Kelley, "Reducing transformer key-value cache size with cross-layer attention," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024, pp. 86 927–86 957.

[26] Z. Wang, F. Liu, P. Xu, Q. Sun, J. Zhao, and L. Jiang, "Evasion: Efficient kv cache compression via product quantization," in *Proceedings of the Design, Automation & Test in Europe (DATE)*, 2025, pp. 1–2.

[27] W. Byun, J. Woo, and S. Mukhopadhyay, "Hessian-aware kv cache quantization for llms," in *Proceedings of the MidWest Symposium on Circuits And Systems (MWSCAS)*, 2024.

[28] J. Wang, Y. Yin, H. Sun, T. Yang, Q. Qi, Z. Zhuang *et al.*, "QoKV: Comprehending and surpassing the hurdles of kv cache quantization," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025.

[29] Z. Liu, X. Luo, J. Guo, W. Ni, Y. Zhou, Y. Guan *et al.*, "VQ-LLM: High-performance code generation for vector quantization augmented llm inference," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025.

[30] J. Ou, Y. Chen, S. Jiang, and W. Tian, "Enhancing large language model inference efficiency via lookahead cache filtering," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025.

[31] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019, pp. 3519–3529.

[32] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[33] D. Busbridge, J. Ramapuram, P. Ablin, T. Likhomanenko, E. G. Dhekane, X. Suau Cuadros *et al.*, "How to scale your ema," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 73 122–73 174.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez *et al.*, "Attention is all you need," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[35] M. Roemmele, C. A. Bejan, and A. S. Gordon, "Choice of plausible alternatives: An evaluation of commonsense causal reasoning." in *AAAI spring symposium: logical formalizations of commonsense reasoning*, 2011, pp. 90–95.

[36] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018, pp. 2381–2391.

[37] S. Narayan, S. B. Cohen, and M. Lapata, "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018, pp. 1797–1807.

[38] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," pp. 1073–1083, 2017.

[39] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024, pp. 3119–3137.

[40] R. Truman, "Searching for needles in a haystack," *Journal of Creation*, vol. 20, no. 2, pp. 90–99, 2006.

[41] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," *arXiv preprint arXiv:2401.02385*, 2024.

[42] A. Yang, B. Yu, C. Li, D. Liu, F. Huang, H. Huang *et al.*, "Qwen2.5-1m technical report," *arXiv preprint arXiv:2501.15383*, 2025.

[43] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.

[44] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 38–45.

Authorized licensed use limited to: Hohai University Library. Downloaded on July 21,2026 at 03:29:46 UTC from IEEE Xplore. Restrictions apply.