KVRECON: KV CACHE MERGING VIA RECONSTRUCTION ERROR MINIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly rely on long contexts, making the memory footprint of key-value (KV) caches a major bottleneck for efficient inference and motivating the need for effective KV cache compression. Prior KV cache merging methods rely on predefined criteria for selecting entries to merge, or heuristic merging operations to combine them, which can introduce substantial discrepancies in model predictions after compression. In this paper, we introduce KVRecon, a KV cache merging method that minimizes the reconstruction error in the output of the attention module. It groups KV entries into multiple sets, and derives merged key and value representations for each set to preserve the output after merging. We further identify KV entries whose attention logits or value representations differ substantially from those of their corresponding merged entries to prevent inaccurate merging. Extensive experiments across multiple LLMs and downstream tasks demonstrate that KVRecon consistently outperforms existing KV cache compression methods.

1 INTRODUCTION

Large language models (LLMs) have emerged as the standard backbone for a wide range of language processing tasks (e.g., question answering (Brown et al., 2020), document summarization (Zhang et al., 2024a), reasoning (Kojima et al., 2022)). These models have recently been extended to long-context scenarios, where KV caching stores the key-value (KV) entries of previously generated tokens and reuses them during autoregressive generation to avoid redundant computation. However, the memory footprint of the cache grows linearly with context length, posing a significant bottleneck for efficient long-context inference. Consequently, KV cache compression techniques have gained significant attention to improve inference efficiency.

Existing KV cache compression approaches include eviction-based methods and merging-based ones. Eviction-based methods estimate the importance of individual KV entries using predefined metrics (e.g., attention scores (Li et al., 2024; Kim et al., 2025), token position (Xiao et al., 2024)), and retain only those with high importance scores. However, directly discarding the remaining KV entries inevitably leads to information loss. Merging-based methods, on the other hand, aggregate similar KV entries to reduce the cache size rather than simply removing them. These methods typically determine which KV entries to merge and how to combine them into fewer entries. They often rely on predefined selection criteria (Cui & Xu, 2025; Tian et al., 2026) or heuristic merging operations (Zhang et al., 2024b; Wan et al., 2025), leading to substantial discrepancies in model predictions after compression (Fig. 1a).

In this paper, we introduce KVRecon, a KV cache merging method that explicitly minimizes the reconstruction error in the output of the attention module during inference (Fig. 1b). Specifically, we group KV entries into multiple sets, and derive merged key and value representations for each set to minimize the reconstruction error. However, under highly constrained cache budgets, a large number of dissimilar KV entries are forced to be grouped together, making it difficult for them to be accurately represented with a single merged entry. To address this issue, we introduce merging conditions that determine whether each KV entry can be safely merged based on the differences in attention logits and value representations between the original and merged entries. We merge only the KV entries satisfying the conditions to prevent inaccurate merging. Extensive experiments across various downstream tasks demonstrate the effectiveness of KVRecon.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

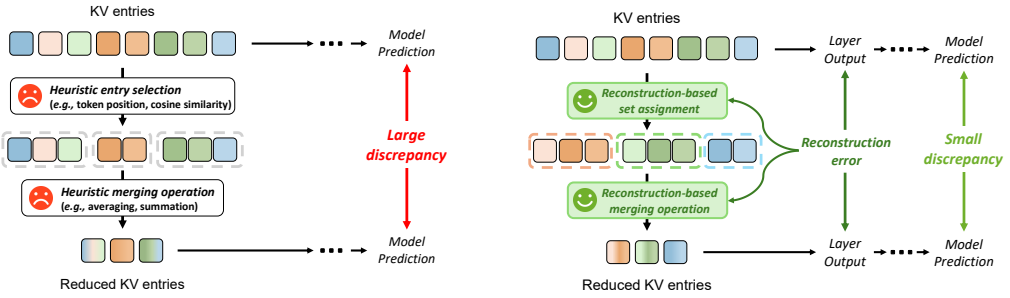


Figure 1: Visual comparison of conventional KV cache merging methods and KVRecon. (a) Existing merging methods rely on predefined criteria to select KV entries for merging or heuristic merging operations to combine the selected entries. (b) KVRecon groups KV entries into multiple sets to represent each set with a single merged entry, and derives the corresponding merged representation that minimizes the reconstruction error in the output of the attention module.

Our contributions are summarized as follows:

- We formulate KV cache merging as a reconstruction problem for the output of the attention module, and derive both the set assignment and the corresponding merged key/value representations accordingly.
- We introduce conditions that identify KV entries unsuitable for merging based on the differences in attention logits and value representations between the original and corresponding merged entries, to prevent inaccurate merging.
- We achieve state-of-the-art performance among KV cache compression methods, with consistent improvements on various downstream tasks.

2 Related Works

KV cache eviction. Eviction-based methods retain a subset of KV entries and discard the remaining ones to reduce the cache size. StreamingLLM (Xiao et al., 2024) is an early approach that retains a small number of initial KV entries and the most recent ones. SnapKV (Li et al., 2024) estimates the importance of KV entries using attention scores obtained from an observation window, and PyramidKV (Cai et al., 2024) allocates different cache budgets across layers according to their attention patterns. TOVA (Oren et al., 2024) evicts KV entries with low attention scores from the current query whenever the cache exceeds the target budget. Since the retained KV entries depend on immediate queries, the compressed cache may be less suitable for reuse with future queries.

KVzip (Kim et al., 2025) constructs a sequence consisting of a repeat prompt followed by the original context, and estimates the importance of each KV entry based on its maximum attention score over the sequence, allowing the compressed cache to be reused across subsequent queries. ReST-KV (An et al., 2026) measures the reconstruction error after removing each KV entry, and uses it as an importance score for eviction. Our method also considers reconstruction error during compression, but uses it to explicitly derive merged key and value representations that minimize discrepancies in the output of the attention module.

KV cache merging. Merging-based methods combine multiple KV entries into a single entry to reduce the cache size rather than simply discarding them. For instance, CaM (Zhang et al., 2024b) selects KV entries with low importance scores and merges them into nearby retained entries by adding equal fractions of their value vectors to those entries. D_2O (Wan et al., 2025) selectively merges KV entries with high cosine similarity between their keys. CentroidKV (Hu et al., 2025) groups KV entries according to the cosine similarity between their keys, and merges the entries within each group. However, these methods rely on predefined criteria for selecting and merging KV entries without explicitly minimizing the reconstruction error in the output of the attention module.

Similar to ours, the works of (Cui & Xu, 2025; Tian et al., 2026) explicitly consider the error introduced by merging. AsymKV (Cui & Xu, 2025) merges adjacent keys based on the curvature of the language modeling loss, and combines their corresponding values by summation. KeepKV (Tian et al., 2026) merges two KV entries at a time, and derives the corresponding merged key and value representations to minimize the resulting output perturbation. In contrast, KVRecon determines both the set assignment and the merged key and value representations for each set to minimize the reconstruction error in the output of the attention module. To prevent inaccurate merging, KVRecon further identifies and excludes KV entries whose attention logits or value representations differ significantly from those of their corresponding merged entries.

3 METHOD

In this section, we first provide a formulation of the KV cache merging problem (Sec. 3.1). We then introduce our reconstruction-based merging method for a given set of KV entries (Sec. 3.2.1), and present conditions for determining whether the entries can be safely merged without significant reconstruction error (Sec. 3.2.2).

3.1 PROBLEM SETUP

Attention mechanism with KV cache. Autoregressive transformers exploit KV cache to reuse contextual information from previously processed tokens. Consider a sequence of query–key–value triplets $\{(\mathbf{q}_n, \mathbf{k}_n, \mathbf{v}_n)\}_{n=1}^{N}$ fed into the attention module, along with a set of cached KV entries $\{(\tilde{\mathbf{k}}_i, \tilde{\mathbf{v}}_i)\}_{i=1}^{S}$, where we denote N and S as the input sequence length and the number of cached KV entries, respectively. The output of the attention module at n-th time step is computed as follows:

$$\mathbf{o}_n = \sum_{j=1}^{n} \frac{\exp(\mathbf{q}_n^\top \mathbf{k}_j)}{Z_n} \mathbf{v}_j + \sum_{i=1}^{S} \frac{\exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i)}{Z_n} \tilde{\mathbf{v}}_i, \tag{1}$$

where the normalization term Z_n is given by

$$Z_n = \sum_{j=1}^{n} \exp(\mathbf{q}_n^\top \mathbf{k}_j) + \sum_{i=1}^{S} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i). \tag{2}$$

Note that we omit the standard $1/\sqrt{d}$ scaling in all query-key dot products for simplicity, where we denote d as the dimensionality of the query and key representations. Our goal is to merge S cached KV entries $\{(\tilde{\mathbf{k}}_i, \tilde{\mathbf{v}}_i)\}_{i=1}^{S}$ into T compressed entries $\{(\hat{\mathbf{k}}_t, \hat{\mathbf{v}}_t)\}_{t=1}^{T}$ ($T < S$) with minimal perturbation to the attention module's output for arbitrary input sequences. After merging, Eq. 1 becomes

$$\hat{\mathbf{o}}_n = \sum_{j=1}^{n} \frac{\exp(\mathbf{q}_n^\top \mathbf{k}_j)}{\hat{Z}_n} \mathbf{v}_j + \sum_{t=1}^{T} \frac{\exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t)}{\hat{Z}_n} \hat{\mathbf{v}}_t, \tag{3}$$

$$\hat{Z}_n = \sum_{j=1}^{n} \exp(\mathbf{q}_n^\top \mathbf{k}_j) + \sum_{t=1}^{T} \exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t). \tag{4}$$

We introduce an additive bias b_t to the attention logit of each compressed KV entry, following prior works (Bolya et al., 2023; Tian et al., 2026).

Reconstruction error minimization. To minimize the discrepancy between the attention outputs before and after merging for arbitrary input sequences, i.e., $\hat{\mathbf{o}}_n \approx \mathbf{o}_n$, we first impose the following condition on the part of the output computed from the input sequence:

$$\sum_{j=1}^{n} \frac{\exp(\mathbf{q}_n^\top \mathbf{k}_j)}{Z_n} \mathbf{v}_j \approx \sum_{j=1}^{n} \frac{\exp(\mathbf{q}_n^\top \mathbf{k}_j)}{\hat{Z}_n} \mathbf{v}_j. \tag{5}$$

To satisfy Eq. 5 for arbitrary $\{(\mathbf{k}_j, \mathbf{v}_j)\}_{j=1}^{n}$, the normalization terms before and after merging should be close, i.e., $\hat{Z}_n \approx Z_n$, which reduces to:

$$\sum_{t=1}^{T} \exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t) \approx \sum_{i=1}^{S} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i). \tag{6}$$

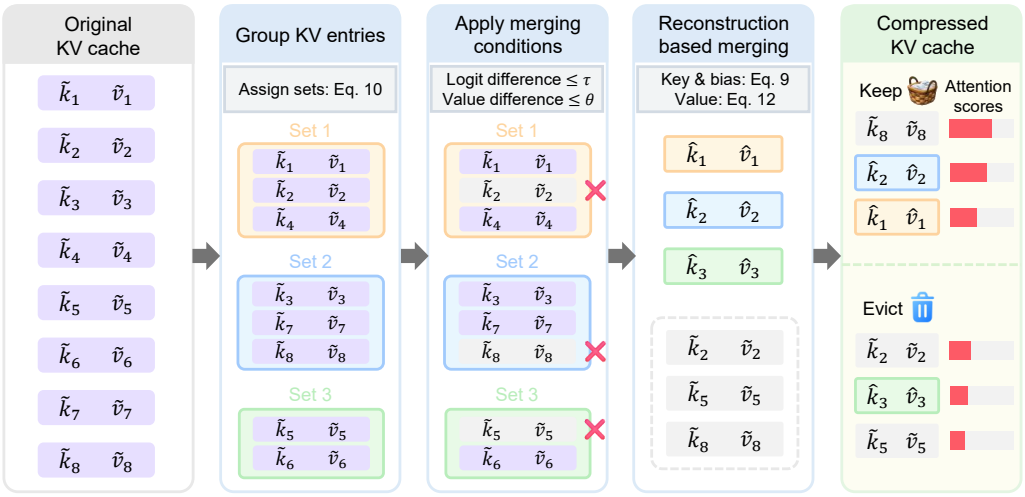


Figure 2: Visualization of KVRecon for $S = 8$ and $T = 3$. We first group KV entries into multiple sets using Eq. 10, and identify the entries that satisfy the merging conditions in Eq. 14 and Eq. 18. We combine the selected entries in each set into a single representation using Eq. 9 and Eq. 12. We then evict entries with low attention scores to meet the target cache budget.

Given $\hat{Z}_n \approx Z_n$, preserving the attention output further requires

$$\sum_{t=1}^{T} \exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t)\hat{\mathbf{v}}_t \approx \sum_{i=1}^{S} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i)\tilde{\mathbf{v}}_i. \tag{7}$$

We first determine $\{(\hat{\mathbf{k}}_t, b_t)\}_{t=1}^T$ to satisfy Eq. 6, and then obtain $\{\hat{\mathbf{v}}_t\}_{t=1}^T$ to satisfy Eq. 7 for arbitrary input queries $\mathbf{q}_n$. In practice, following KVzip (Kim et al., 2025), we construct a sequence consisting of a repeat prompt followed by the original context, and sample N tokens from the sequence to obtain the input query–key–value triplets. Note that the sequence depends only on the prefilled context, and does not use any test-time query.

3.2 KVRecon

In this subsection, we present KVRecon, which groups KV entries into multiple sets, and derives merged key and value representations for each set to minimize the reconstruction error. We additionally identify and exclude KV entries that cannot be accurately represented with their assigned merged entries. We describe the reconstruction-based merging in Sec. 3.2.1, and the merging conditions in Sec. 3.2.2. A visualization of KVRecon is illustrated in Fig. 2.

3.2.1 Reconstruction-based merging

Merged key representation. We first search for $\{(\hat{\mathbf{k}}_t, b_t)\}_{t=1}^T$ that satisfy Eq. 6. We decompose Eq. 6 by grouping the cached KV entries into T sets and consider each set separately as follows:

$$\exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t) \approx \sum_{i \in \mathcal{C}_t} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i), \quad t = 1, \ldots, T, \tag{8}$$

where we denote $\mathcal{C}_t$ as the set of KV entries merged into $(\hat{\mathbf{k}}_t, \hat{\mathbf{v}}_t)$. For each set $\mathcal{C}_t$, we derive the merged key and bias for approximating Eq. 8, as presented in Proposition 1.

Proposition 1. For a given set $\mathcal{C}_t$, the approximation in Eq. 8 is obtained by setting

$$\hat{\mathbf{k}}_t = \frac{1}{|\mathcal{C}_t|} \sum_{i \in \mathcal{C}_t} \tilde{\mathbf{k}}_i, \qquad b_t = \log |\mathcal{C}_t|, \tag{9}$$

assuming that $\exp(\mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)) \approx 1 + \mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)$.

We provide the derivation in Appendix A.1. We initialize $\{(\hat{\mathbf{k}}_t, \hat{\mathbf{v}}_t)\}_{t=1}^{T}$ by uniformly sampling from the cached KV entries along the temporal dimension. Based on Proposition 1, each cached KV entry is assigned to the set that minimizes the difference between the attention logits of the original and merged entries as follows:

$$\mathcal{C}_t = \left\{ i \;\middle|\; t = \arg\min_j \mathbb{E}_n \left[\left| \mathbf{q}_n^\top (\hat{\mathbf{k}}_j - \tilde{\mathbf{k}}_i) \right| \right] \right\}, \quad t = 1, \ldots, T, \tag{10}$$

where we denote by $\mathbb{E}_n[\cdot]$ the expectation over the input query representations.

Merged value representation. After grouping the cached KV entries into T sets using Eq. 10, we determine the merged value $\hat{\mathbf{v}}_t$ for each set. For each set $\mathcal{C}_t$, we require

$$\exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t)\hat{\mathbf{v}}_t \approx \sum_{i \in \mathcal{C}_t} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i)\tilde{\mathbf{v}}_i, \quad t = 1, \ldots, T. \tag{11}$$

Proposition 2. Given $\hat{\mathbf{k}}_t$ and b_t from Proposition 1, the merged value that minimizes the error in Eq. 11 is given by

$$\hat{\mathbf{v}}_t = \frac{\sum_{n=1}^{N} a_{n,t}\mathbf{r}_{n,t}}{\sum_{n=1}^{N} a_{n,t}^2}, \tag{12}$$

where

$$a_{n,t} = \sum_{i \in \mathcal{C}_t} \alpha_{n,i}, \qquad \mathbf{r}_{n,t} = \sum_{i \in \mathcal{C}_t} \alpha_{n,i}\tilde{\mathbf{v}}_i, \tag{13}$$

and $\alpha_{n,i} = \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i)/Z_n$ is the attention score between $\mathbf{q}_n$ and $\tilde{\mathbf{k}}_i$.

The derivation is provided in Appendix A.2. After assigning KV entries to each set $\mathcal{C}_t$ using Eq. 10, we use Eq. 9 and Eq. 12 to obtain the corresponding merged KV entry.

3.2.2 Merging conditions

We derive in Sec. 3.2.1 a merged KV entry with minimal reconstruction error for a given set of KV entries. Nevertheless, a single merged entry may not accurately represent all KV entries in a set, particularly under highly constrained cache budgets, where a large number of dissimilar KV entries are forced to be grouped together. To this end, we introduce conditions for determining whether each KV entry can be safely merged without introducing substantial error.

First, the approximation used in Proposition 1 becomes less accurate as the difference between the attention logits of the original and merged entries increases. We thus impose the following condition:

$$\mathbb{E}_n \left[\left| \mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t) \right| \right] \leq \tau, \quad \forall i \in \mathcal{C}_t, \tag{14}$$

where τ controls the allowable difference between attention logits of the original and merged entries.

Second, the approximation of the value term in Eq. 11 can be expressed as

$$\sum_{i \in \mathcal{C}_t} \exp(\mathbf{q}_n^\top \tilde{\mathbf{k}}_i)\tilde{\mathbf{v}}_i = \exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t) \sum_{i \in \mathcal{C}_t} \exp\left(\mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)\right) \tilde{\mathbf{v}}_i \tag{15}$$

$$\approx \exp(\mathbf{q}_n^\top \hat{\mathbf{k}}_t + b_t)\hat{\mathbf{v}}_t. \tag{16}$$

Using $b_t = \log |\mathcal{C}_t|$, Eq. 15 yields

$$\hat{\mathbf{v}}_t \approx \frac{1}{|\mathcal{C}_t|} \sum_{i \in \mathcal{C}_t} \exp\left(\mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)\right) \tilde{\mathbf{v}}_i. \tag{17}$$

Accordingly, we impose the second condition as

$$\mathbb{E}_n \left[\left\| \exp\left(\mathbf{q}_n^\top (\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)\right) \tilde{\mathbf{v}}_i - \hat{\mathbf{v}}_t \right\|_2 \right] \leq \theta, \quad \forall i \in \mathcal{C}_t, \tag{18}$$

where θ controls the allowable difference in the value representation. We merge a KV entry into its assigned set only when both conditions in Eq. 14 and Eq. 18 are satisfied. KV entries that do not satisfy either condition are retained without merging. Consequently, the resulting cache may exceed the target budget T, and we further evict KV entries with low attention scores to meet the budget. Note that we determine τ and θ offline using a small calibration set, as detailed in Appendix B. We show the overall procedure of KVRecon in Algorithm 1.

Algorithm 1 KVRecon

Input: KV cache $\{(\tilde{\mathbf{k}}_i, \tilde{\mathbf{v}}_i)\}_{i=1}^{S}$; target cache budget T
Hyperparameters: Number of input queries N; number of merging rounds R
Collect input query representations $\{\mathbf{q}_n\}_{n=1}^{N}$.
Define $\rho(t) = \lceil 1 + (t-1)(S-1)/(T-1) \rfloor$.
Initialize $(\hat{\mathbf{k}}_t, \hat{\mathbf{v}}_t) = (\tilde{\mathbf{k}}_{\rho(t)}, \tilde{\mathbf{v}}_{\rho(t)})$ and $b_t = 0$ for $t = 1, \ldots, T$.
Initialize $\mathcal{U}$ with all cached KV entries.
for $r = 1, \ldots, R$ **do**
    Assign each KV entry in $\mathcal{U}$ to a set using Eq. 10.
    Identify KV entries satisfying Eq. 14 and Eq. 18.
    Exclude the remaining KV entries from each set $\mathcal{C}_t$.
    Update $\{(\hat{\mathbf{k}}_t, \hat{\mathbf{v}}_t)\}_{t=1}^{T}$ using Eq. 9 and Eq. 12.
    Set $b_t = \log |\mathcal{C}_t|$ for $t = 1, \ldots, T$.
    Update $\mathcal{U}$ with the excluded KV entries.
end for
Evict KV entries with low attention scores until the cache size reaches T.
Output: Compressed KV cache

4 EXPERIMENTS

4.1 IMPLEMENTATION DETAILS

Baselines and models. We compare KVRecon with state-of-the-art KV cache compression methods, including eviction-based methods, StreamingLLM (Xiao et al., 2024) and KVzip (Kim et al., 2025), and merging-based methods, D_2O (Wan et al., 2025) and AsymKV (Cui & Xu, 2025). We evaluate these methods on Llama3.1-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), and Qwen2.5-7B (Bai et al., 2023a).

Benchmarks and evaluation protocol. We evaluate KVRecon on LongBench (Bai et al., 2023b) and SCBench (Li et al., 2025). LongBench evaluates the long-context understanding capabilities of LLMs across six categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. Following prior works (Wan et al., 2025; Cui & Xu, 2025), we evaluate 16 tasks in LongBench. Each context is paired with a single query or task instruction, with context lengths of up to 18.4K tokens. We use the official evaluation metric for each task, and report the average score across all tasks.

SCBench evaluates scenarios in which a long context is shared across multiple queries. Following KVzip (Kim et al., 2025), we use the preprocessed SCBench released in its official repository, including both shortened and original data. We first evaluate 6 tasks using the shortened data, with context lengths of up to approximately 20.6K tokens, on all three models. We additionally evaluate 9 tasks using the original data, with context lengths of up to approximately 125K tokens, on Llama3.1-8B and Qwen2.5-7B. Note that we evaluate Mistral-7B only on the shortened data, since its context window cannot accommodate the original examples. We compress the KV cache once for each context before observing any query, and reuse the compressed cache across all queries associated with the context. Since the official implementation of D_2O (Wan et al., 2025) uses the evaluation query for cache compression, we modify it to prefill only the shared context and reuse the resulting compressed cache across all queries for a fair comparison.

4.2 RESULTS

Results on LongBench. We show in Table 1 the quantitative comparison of KVRecon with prior KV cache compression methods on 16 tasks in LongBench (Bai et al., 2023b). We find that KVRecon consistently achieves the highest average score among all compared methods, demonstrating the effectiveness of our reconstruction-based approach and merging conditions. It outperforms the strongest baseline by 2.09%p, 2.14%p, and 2.05%p for Llama3.1-8B, Qwen2.5-7B, and Mistral-7B, respectively. In addition, KVRecon outperforms the compared methods on 11 out of 16 tasks for both Llama3.1-8B and Qwen2.5-7B, and on 12 tasks for Mistral-7B. These results demonstrate the effectiveness of KVRecon across diverse long-context tasks, including question answering, summarization, and few-shot learning.

Table 1: Quantitative comparison of various KV cache eviction and merging methods on LongBench (Bai et al., 2023b). We report results on 16 tasks across multiple models at a 10% cache budget.

Methods	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NtrvQA	Qasper	MF-en	Hotpot	2WikiQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PR-en	LCC	RB-P	
Llama-3.1-8B																	
Full cache	31.39	46.69	56.94	58.09	47.77	31.33	34.60	25.43	26.96	73.00	91.93	43.71	7.83	100.00	65.04	54.89	49.73
StreamingLLM	20.37	16.65	22.32	35.28	18.64	13.97	23.75	19.07	19.71	46.50	86.39	39.09	2.56	15.50	**61.28**	**62.88**	31.50
D_2O	22.78	15.93	26.17	38.12	19.87	20.31	23.94	20.02	20.86	40.50	89.53	39.68	**8.50**	11.00	57.65	57.55	32.03
AsymKV	25.54	29.94	37.24	47.12	33.70	25.22	27.93	22.73	25.83	61.00	90.00	**41.95**	4.06	60.50	59.89	48.23	40.05
KVZip	27.42	40.51	**55.39**	49.97	38.52	25.10	31.22	24.00	25.20	55.00	89.18	39.82	5.75	64.00	53.44	60.13	42.79
KVRecon	**31.16**	**41.70**	54.37	**54.96**	**40.95**	**25.96**	**31.82**	**25.24**	**26.19**	**64.00**	**90.13**	41.79	6.33	**67.50**	56.99	60.98	**44.88**
Qwen2.5-7B																	
Full cache	29.17	47.14	51.51	60.40	53.95	32.92	33.37	23.39	24.39	76.50	84.87	44.72	8.00	100.00	47.86	41.31	47.47
StreamingLLM	16.43	17.47	20.68	33.19	31.73	13.39	24.44	18.15	16.23	33.50	78.80	37.93	4.00	16.00	32.09	39.25	27.08
D_2O	0.45	0.00	0.10	0.00	0.00	0.00	0.00	0.84	1.00	0.00	0.35	1.19	0.00	0.00	11.00	11.17	1.63
AsymKV	23.90	23.26	31.13	40.05	30.24	14.54	30.74	20.85	21.74	65.00	80.69	**43.25**	3.50	35.50	30.06	32.99	34.24
KVZip	**26.41**	33.03	45.10	51.01	**49.39**	22.62	31.12	23.05	20.08	53.50	**83.56**	42.05	4.00	51.50	34.31	**41.41**	38.26
KVRecon	26.10	**35.14**	**45.74**	**52.37**	46.25	**27.58**	**31.35**	**23.31**	**22.40**	**66.00**	81.02	42.39	**5.00**	**61.00**	**37.19**	39.92	**40.40**
Mistral-7B																	
Full cache	26.98	37.83	51.11	50.64	38.11	27.87	34.50	25.80	26.62	75.50	89.76	47.26	6.00	94.50	68.17	67.39	48.00
StreamingLLM	17.66	13.92	23.07	30.10	20.25	12.22	24.50	19.20	17.31	52.50	85.63	42.87	**7.14**	14.50	59.39	60.35	31.29
D_2O	20.25	13.57	25.12	36.51	22.99	13.16	25.24	19.11	20.74	48.50	87.67	43.45	4.41	7.00	57.56	59.21	31.53
AsymKV	22.24	24.44	32.61	44.71	27.83	16.63	28.82	22.50	25.58	67.00	87.72	43.98	2.63	54.00	62.59	58.57	38.87
KVZip	22.70	37.13	49.03	50.23	**38.78**	20.77	34.51	23.64	25.75	58.00	88.35	44.01	4.33	85.50	**64.17**	**63.08**	44.37
KVRecon	**25.66**	**37.51**	**50.13**	**50.42**	37.78	**24.74**	**34.66**	**25.47**	**25.94**	**74.00**	**88.95**	**45.20**	5.00	**91.00**	63.24	63.05	**46.42**

Table 2: Quantitative comparison of various KV cache compression methods on 6 tasks sampled from a shortened variant of SCBench (Li et al., 2025) released in (Kim et al., 2025). We report results for various models at a 10% cache budget.

Methods	Retr.KV	Retr.Prefix-Suffix	Code.RepoQA	ICL.ManyShot	En.Summary	Math.Find	Avg.
Llama-3.1-8B							
Full cache	89.2	93.0	55.5	35.6	42.5	36.7	58.8
StreamingLLM	7.9	9.4	1.8	24.1	34.6	5.3	13.9
D_2O	3.1	4.6	2.7	30.4	39.1	23.7	17.3
AsymKV	0.5	2.6	1.8	32.6	37.1	22.7	16.2
KVZip	28.7	46.0	35.5	31.9	**42.4**	16.5	33.5
KVRecon	**69.7**	**63.4**	**43.6**	**33.3**	41.9	**24.8**	**46.1**
Qwen2.5-7B							
Full cache	85.5	88.8	72.7	35.6	39.2	37.3	59.9
StreamingLLM	7.0	8.2	2.7	33.7	30.8	10.8	15.5
D_2O	0.2	0.0	3.6	33.7	38.2	13.0	14.8
AsymKV	0.0	0.4	4.6	**34.4**	34.8	2.8	12.8
KVZip	29.0	0.6	37.3	31.1	39.2	**35.8**	28.8
KVRecon	**47.6**	**13.8**	**52.7**	**34.4**	**39.9**	33.2	**36.9**
Mistral-7B							
Full cache	59.2	72.0	57.3	31.9	41.0	36.8	49.7
StreamingLLM	4.0	6.8	3.6	27.8	35.7	7.7	14.3
D_2O	0.0	4.0	0.0	31.9	40.3	25.3	16.9
AsymKV	2.7	3.6	0.9	**32.2**	37.9	31.3	18.1
KVZip	16.1	15.2	30.0	31.9	43.0	**32.0**	28.0
KVRecon	**52.5**	**63.8**	**45.5**	29.3	**43.9**	31.2	**44.4**

Results on SCBench. We show in Table 2 the quantitative comparison of KVRecon with prior KV cache compression methods on the shortened SCBench data released in the official repository of (Kim et al., 2025). We find that KVRecon achieves the highest average score across all three models, demonstrating its effectiveness in multi-query scenarios. In particular, KVRecon substantially outperforms the compared methods on the tasks of Retr.KV, Retr.Prefix-Suffix, and Code.RepoQA. For example, KVRecon outperforms KVzip (Kim et al., 2025) by 41.0%p on Retr.KV with Llama3.1-8B, and by 48.6%p on Retr.Prefix-Suffix with Mistral-7B. We also observe that the merging methods, i.e., D_2O (Wan et al., 2025) and AsymKV (Cui & Xu, 2025), incur substantial performance degradation compared to KVRecon on these tasks. We additionally report results on the original SCBench data in Appendix C.

Table 3: Quantitative results for applying different merged representations for keys and values in each set. We compare Eq. 9 and Eq. 12 with alternative merged representations of keys and values, respectively. We report the performance on Retr.KV using the shortened SCBench (Li et al., 2025) released in (Kim et al., 2025) at a 10% cache budget. Note that we apply the merging conditions defined by Eq. 14 and Eq. 18 in this analysis.

Merged key	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
D_2O	53.6	30.9	38.3
AsymKV	38.8	36.0	33.4
Eq. 9	**69.7**	**47.6**	**52.5**

(a) Merged key representation

Merged value	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
D_2O	58.9	44.2	28.4
AsymKV	67.8	47.2	52.3
Eq. 12	**69.7**	**47.6**	**52.5**

(b) Merged value representation

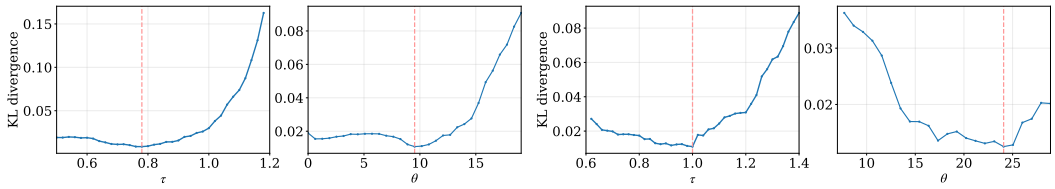
Table 4: Ablation analysis for the merging conditions in Sec. 3.2.2. We report the performances of TREC in LongBench (Bai et al., 2023b) and Retr.KV in shortened SCBench (Li et al., 2025) at a 10% cache budget. Note that we use the merged representations in Eq. 9 and Eq. 12 for keys and values in this analysis, respectively.

Eq. 14	Eq. 18	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
		37.5	6.5	57.5
✓		61.5	64.5	72.5
	✓	63.0	62.0	**74.0**
✓	✓	**64.0**	**66.0**	**74.0**

(a) TREC

Eq. 14	Eq. 18	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
		0.0	0.0	0.0
✓		61.3	46.5	52.0
	✓	68.4	45.4	46.9
✓	✓	**69.7**	**47.6**	**52.5**

(b) Retr.KV



(a) Llama3.1-8B (b) Mistral-7B

Figure 3: KL divergence between the output distributions of the full and compressed models for (a) Llama3.1-8B and (b) Mistral-7B under varying τ (left) and θ (right). We first generate an answer before compression, and then teacher-force the same answer for all compression methods. We report the KL divergence averaged over the generated token positions. For the left plots, we do not apply Eq. 18; for the right plots, we do not apply Eq. 14. The red dashed lines indicate the values that minimize the KL divergence.

4.3 Discussion

Analysis on merged key/value representations. We present in Table 3 the results of applying different merged representations for keys and values. We compare Eq. 9 and Eq. 12 with the corresponding merged representations used in D_2O (Wan et al., 2025) and AsymKV (Cui & Xu, 2025). We observe that the proposed key and value representations consistently achieve the best performance across models, indicating that they better preserve model performance after KV cache compression than the representations used in prior methods.

Analysis on merging conditions. We present in Table 4 an ablation analysis of the merging conditions defined in Sec. 3.2.2. We can see that applying either Eq. 14 or Eq. 18 substantially improves performance in both single-query and multi-query settings, demonstrating that each condition effectively prevents inaccurate merging. We further observe that applying both conditions generally achieves the best performance across models and tasks, indicating that the two conditions complement each other in reducing perturbations caused by merging.

We show in Fig. 3 the KL divergence between the output distributions of the original and compressed models for different values of τ and θ. We observe that large values of τ and θ substantially increase the KL divergence for both models, indicating that allowing more dissimilar KV entries to be merged introduces larger output perturbations. In addition, we find that small values of τ and θ also increase the KL divergence for both models, suggesting that stricter merging conditions limit the number of merged KV entries and increase the reliance on eviction.

Reconstruction error. We show in Fig. 4a the reconstruction error of a particular attention module after applying various KV cache compression methods. We find that KVRecon achieves the lowest reconstruction error among the compared methods, demonstrating that it better preserves the output of the attention module after compression. We also observe that the reconstruction errors of the

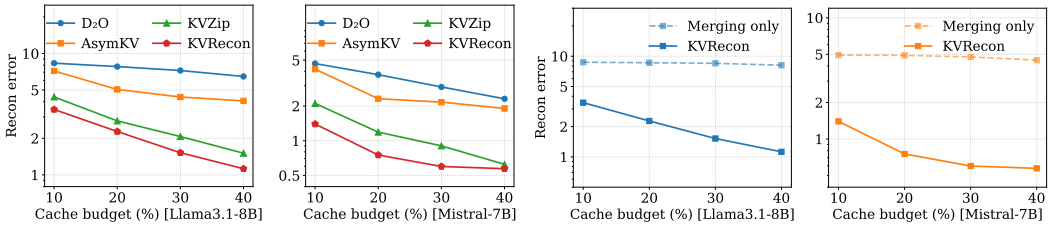
Table 5: Quantitative comparison of different set assignment techniques. We compare Eq. 10 with alternative assignment techniques, including token position (Cui & Xu, 2025), cosine similarity between keys (Wan et al., 2025; Hu et al., 2025; Tian et al., 2026), and value difference in Eq. 18. We report the results across multiple models and tasks at a 10% cache budget.

Set assignment	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
Adjacent position	56.0	53.5	53.5
Key cosine similarity	61.5	**66.0**	73.0
Value difference in Eq. 18	61.5	56.5	72.5
Logit difference in Eq. 10	**64.0**	**66.0**	**74.0**

(a) TREC

Set assignment	Llama3.1-8B	Qwen2.5-7B	Mistral-7B
Adjacent position	32.8	32.8	33.0
Key cosine similarity	48.7	44.0	20.4
Value difference in Eq. 18	42.5	46.5	39.0
Logit difference in Eq. 10	**69.7**	**47.6**	**52.5**

(b) Retr.KV



(a) (b)

Figure 4: (a) Reconstruction error for a particular attention module of Llama3.1-8B (Grattafiori et al., 2024) and Mistral-7B (Jiang et al., 2023), compared across cache budgets and compression methods. (b) Reconstruction error in the same module of Llama3.1-8B and Mistral-7B using KVRecon with and without the merging conditions in Eq. 14 and Eq. 18. We can see that KVRecon achieves the lowest reconstruction error among the compared methods, and the error increases substantially if the merging conditions are not applied.

merging-based methods, i.e., AsymKV (Cui & Xu, 2025) and D_2O (Wan et al., 2025), remain higher than that of KVRecon even at larger cache budgets. This indicates that their merging methods introduce considerable perturbations to the output of the attention module.

In addition, we compare in Fig. 4b the reconstruction error with and without the merging conditions in Sec. 3.2.2. We can see that the merging conditions significantly reduce the reconstruction error across all models and cache budgets. This indicates that applying Eq. 9 and Eq. 12 to dissimilar KV entries can introduce substantial reconstruction error. We also observe that, without the merging conditions, the reconstruction error remains nearly unchanged as the cache budget increases. This suggests that increasing the cache budget does not prevent inaccurate merging when relying solely on Eq. 9 and Eq. 12.

Set assignment. We show in Table 5 the quantitative results obtained with different set assignment techniques. We first consider a similarity-based assignment used in prior merging methods (Wan et al., 2025; Hu et al., 2025; Tian et al., 2026), which identify KV entries to merge based on the cosine similarity between their keys. We also consider a position-based assignment used in (Cui & Xu, 2025), which exploits the local similarity of adjacent keys, as well as an assignment based on the value difference in Eq. 18. We observe that the position-based assignment incurs substantial performance degradation, indicating that similar KV entries are not necessarily located at adjacent positions. We also find that Eq. 10 consistently achieves the best performance across models and tasks, demonstrating the effectiveness of our set assignment technique.

5 Conclusion

In this paper, we introduced KVRecon, a KV cache merging method that minimizes the reconstruction error in the output of the attention module. We derive merged key and value representations based on the reconstruction error, and introduce merging conditions to prevent KV entries that cannot be accurately merged from being combined into a single representation. Extensive experiments on diverse downstream tasks demonstrate that KVRecon consistently outperforms existing KV cache compression methods in both single-query and multi-query settings. Our analyses further demonstrate the effectiveness of the merged key and value representations and merging conditions in KVRecon.

AI USE STATEMENT

In this work, we used generative AI tools only to polish the writing. We have not used generative AI tools for idea generation, retrieval of related works, or experimental design. All AI-assisted text was reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the complete algorithm of KVRecon in Sec. 3.2 and Algorithm 1. Detailed derivations of the merged key and value representations are provided in Appendix A, and the procedure for hyperparameter calibration is described in Appendix B. We describe the implementation details in Sec. 4.1, including models, benchmarks, and evaluation protocols. All models and benchmarks we used are publicly available online.

REFERENCES

Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. Rest-kv: Robust kv cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. *ICLR*, 2026.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv*, 2023a.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *arXiv*, 2023b.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *ICLR*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv*, 2024.

Wanyun Cui and Mingwei Xu. Homogeneous keys, heterogeneous values: Exploiting local kv cache asymmetry for long-context llms. *NeurIPS*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv*, 2024.

Jie Hu, Shengnan Wang, Yutong He, Ping Gong, Jiawei Yi, Juncheng Zhang, Youhui Bai, Renhai Chen, Gong Zhang, Cheng Li, et al. Centroidkv: Efficient long-context llm inference via kv cache clustering. *TMLR*, 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, et al. Mistral 7b. *arXiv*, 2023.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *NeurIPS*, 2025.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *NeurIPS*, 2022.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. Scbench: A kv cache-centric analysis of long-context methods. In *ICLR*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *NeurIPS*, 2024.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. Keepkv: Achieving periodic lossless kv cache compression for efficient llm inference. *AAAI*, 2026.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, et al. D2o: Dynamic discriminative operations for efficient long-context inference of large language models. In *ICLR*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.

Yang Zhang, Hanlei Jin, Dan Meng, Jun Wang, and Jinghua Tan. A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods. *arXiv*, 2024a.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *ICML*, 2024b.

A Derivation of merged key/value representations

A.1 Derivation of proposition 1

We take the logarithm of Eq. 8 and apply a first-order Taylor approximation w.r.t. $\mathbf{q}_n^\top(\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)$ as follows:

$$\log\left\{\sum_{i\in\mathcal{C}_t}\exp(\mathbf{q}_n^\top\tilde{\mathbf{k}}_i)\right\} = \mathbf{q}_n^\top\hat{\mathbf{k}}_t + \log\left\{\sum_{i\in\mathcal{C}_t}\exp(\mathbf{q}_n^\top(\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t))\right\} \tag{19}$$

$$\approx \mathbf{q}_n^\top\hat{\mathbf{k}}_t + \log\left\{\sum_{i\in\mathcal{C}_t}\left(1 + \mathbf{q}_n^\top(\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)\right)\right\} \tag{20}$$

$$= \mathbf{q}_n^\top\hat{\mathbf{k}}_t + \log|\mathcal{C}_t| + \log\left\{1 + \frac{1}{|\mathcal{C}_t|}\sum_{i\in\mathcal{C}_t}\mathbf{q}_n^\top(\tilde{\mathbf{k}}_i - \hat{\mathbf{k}}_t)\right\}. \tag{21}$$

The first-order residual term vanishes by setting

$$\hat{\mathbf{k}}_t = \frac{1}{|\mathcal{C}_t|}\sum_{i\in\mathcal{C}_t}\tilde{\mathbf{k}}_i. \tag{22}$$

Eq. 19 then reduces to

$$\log\left\{\sum_{i\in\mathcal{C}_t}\exp(\mathbf{q}_n^\top\tilde{\mathbf{k}}_i)\right\} \approx \mathbf{q}_n^\top\hat{\mathbf{k}}_t + \log|\mathcal{C}_t|, \tag{23}$$

which gives $b_t = \log|\mathcal{C}_t|$, where we denote by $|\mathcal{C}_t|$ the number of KV entries in $\mathcal{C}_t$.

A.2 Derivation of proposition 2

We divide both sides of Eq. 11 by Z_n and search for $\hat{\mathbf{v}}_t$ that minimizes their discrepancy as follows:

$$\arg\min_{\hat{\mathbf{v}}_t}\mathbb{E}_n\left[\left\|\frac{\exp(\mathbf{q}_n^\top\hat{\mathbf{k}}_t + b_t)}{Z_n}\hat{\mathbf{v}}_t - \sum_{i\in\mathcal{C}_t}\alpha_{ni}\tilde{\mathbf{v}}_i\right\|_2^2\right], \tag{24}$$

Using Eq. 8, we approximate

$$\frac{\exp(\mathbf{q}_n^\top\hat{\mathbf{k}}_t + b_t)}{Z_n} \approx \sum_{i\in\mathcal{C}_t}\alpha_{ni}. \tag{25}$$

Accordingly, Eq. 24 reduces to

$$\arg\min_{\hat{\mathbf{v}}_t}\mathbb{E}_n\left[\|a_{nt}\hat{\mathbf{v}}_t - \mathbf{r}_{nt}\|_2^2\right], \tag{26}$$

where

$$a_{nt} = \sum_{i\in\mathcal{C}_t}\alpha_{ni}, \qquad \mathbf{r}_{nt} = \sum_{i\in\mathcal{C}_t}\alpha_{ni}\tilde{\mathbf{v}}_i. \tag{27}$$

Eq. 26 is a least-squares problem with a closed-form solution. The merged value is then given by

$$\hat{\mathbf{v}}_t = \frac{\sum_{n=1}^N a_{nt}\mathbf{r}_{nt}}{\sum_{n=1}^N a_{nt}^2}. \tag{28}$$

B Threshold calibration

We determine τ and θ using an alternating grid search on a small calibration set sampled from the C4 corpus (Raffel et al., 2020). We consider candidate values of τ in $[0, 2]$ with a step size of 0.02 and θ in $[0, 10\nu]$ with a step size of 0.1ν, where we denote ν as the mean L_2-norm of the value tokens in each attention module. Rather than evaluating the entire two-dimensional grid, we alternately optimize τ and θ: we exhaustively search the candidate values of τ with θ fixed, followed by a search over θ with τ fixed. This procedure is repeated for 3 rounds. For each round, we compress the KV cache according to the merging conditions in Eq. 14 and Eq. 18, and select the value that minimizes the KL divergence between the model outputs before and after compression.

Table A: Quantitative comparison of various KV cache compression methods on 9 tasks sampled from the full SCBench (Li et al., 2025) released in (Kim et al., 2025). We report results on Llama-3.1-8B and Qwen2.5-7B under a 20% cache budget.

Methods	Retr.KV	Retr.Prefix-Suffix	Code.RepoQA	ICL.ManyShot	En.Summary	Math.Find	Retr.MultiHop	En.QA	En.MultiChoice	Avg.
Llama-3.1-8B										
Full cache	61.4	26.8	42.7	34.8	39.3	15.5	27.3	46.5	75.0	41.0
StreamingLLM	6.2	6.8	8.4	33.3	29.8	9.7	16.9	20.7	46.8	19.8
AsymKV	0.0	6.0	5.0	31.9	30.4	10.0	18.4	23.9	59.3	20.5
KVZip	0.0	0.6	21.1	**35.2**	37.8	7.8	**33.9**	32.7	59.7	25.4
KVRecon	**50.8**	**16.4**	**29.1**	33.3	**39.1**	**13.5**	27.1	**38.6**	**63.4**	**34.6**
Qwen2.5-7B										
Full cache	67.6	49.0	58.9	37.0	36.9	32.5	40.9	41.3	79.2	49.3
StreamingLLM	14.6	7.4	11.6	34.4	28.6	22.2	13.7	18.9	49.5	22.3
AsymKV	0.0	0.0	9.1	**34.1**	28.3	18.8	17.6	42.1	55.4	22.8
KVZip	35.2	2.6	51.4	33.0	35.3	**35.8**	**40.1**	41.9	**72.2**	38.6
KVRecon	**54.8**	**11.8**	**57.0**	**34.1**	**36.1**	33.5	38.6	**43.9**	**72.2**	**42.4**

C RESULTS ON SCBENCH

We report in Table A the quantitative results of various KV cache compression methods on the original preprocessed SCBench data released in (Kim et al., 2025). We observe that KVRecon achieves the highest average score for both Llama3.1-8B (Grattafiori et al., 2024) and Qwen2.5-7B (Bai et al., 2023a). In addition, we find that KVRecon achieves the highest score on 7 out of 9 tasks for both models, demonstrating consistent performance in extremely long-context settings. Note that we cannot measure the performances of D_2O (Wan et al., 2025) in the original SCBench due to memory limitations.