

ConsistKV: Layer Adaptive KV Cache Compression via Cross Layer Consensus

Dongjie Xu^{1*†}, Kai Qian^{1*}, Weijie Shi², Fenglei Jin³, Jiajie Xu^{1‡}

¹Soochow University, ²The Hong Kong University of Science and Technology, ³Tencent Inc.

Abstract

As LLMs process increasingly long contexts, the KV cache has become a major source of memory and bandwidth overhead during inference. Existing methods compress the KV cache by preserving attention sinks, recent tokens, or layer-local important tokens, but they rarely exploit how token importance is shared and specialized across layers. In this paper, we introduce ConsistKV, an adaptive KV cache compression method that exploits consensus in token importance across layers. For each layer, ConsistKV aggregates importance scores from neighboring layers to identify shared contextual anchors, while using residual local importance to preserve tokens that are salient to the current layer. It further estimates the cache demand of each layer from disagreement between layers and importance dispersion, enabling adaptive budget allocation under a fixed global cache budget. Experiments on LongBench show that ConsistKV achieves the highest average performance among evaluated KV cache compression baselines under the same cache budget.

1 Introduction

Large language models (LLMs) have been increasingly extended to long-context scenarios, enabling applications such as document summarization, question answering, information retrieval, and code generation (Grattafiori et al., 2024; Rozière et al., 2023; Shaham et al., 2023). These applications often require processing long documents or multi-hop evidence (Zhang et al., 2024; Kamaloo et al., 2023). However, during autoregressive decoding, the KV cache grows linearly with the context length and the number of Transformer layers, leading to substantial memory and bandwidth pressure (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2024; Adnan et al., 2024). Therefore, KV

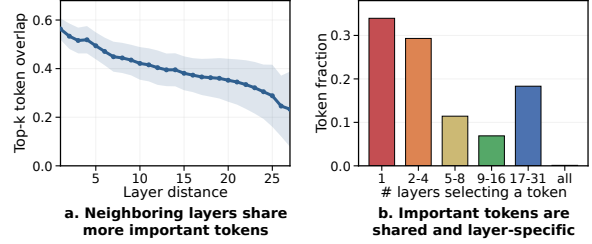


Figure 1: Cross-layer patterns of important tokens.

cache compression has become critical for efficient long-context LLM inference.

Existing KV cache compression methods reduce memory usage by retaining only a subset of cached tokens. StreamingLLM (Xiao et al., 2024) keeps attention sinks and recent tokens for stable streaming generation. Scissorhands (Liu et al., 2023) exploits the persistence of attention importance to evict less influential cached tokens, while H₂O and SnapKV (Zhang et al., 2023; Li et al., 2024) select important tokens according to accumulated attention scores or attention scores from an observation window. Layer-aware methods further assign different cache budgets to different layers to improve cache utilization (Cai et al., 2024; Qin et al., 2025). However, these methods mainly estimate token importance within individual layers, overlooking shared importance across layers and importance unique to each layer. As a result, they may fail to preserve both common contextual anchors and layer dependent information under tight cache budgets.

Motivated by this limitation, we examine whether important tokens exhibit structured patterns across Transformer layers. As shown in Figure 1(a), neighboring layers share a higher fraction of top- k important tokens, and this overlap gradually decreases as the layer distance increases. This suggests that token importance is not independent across layers, but shows clear local consistency. Meanwhile, Figure 1(b) shows that important to-

*Equal contribution

†Email: djxu@stu.suda.edu.cn

‡Corresponding author

kens are only partially shared: some tokens are selected by many layers and act as common contextual anchors, while others are selected by only a few layers and reflect layer specific information. These observations suggest that effective KV cache compression should preserve both consensus tokens shared across layers and diverse tokens that are important to individual layers.

Guided by these findings, we introduce ConsistKV, a KV cache compression method that exploits the cross-layer structure of token importance. Instead of selecting cached tokens independently within each layer, ConsistKV distinguishes tokens consistently emphasized by neighboring layers from those salient only in individual layers. For each layer, ConsistKV first estimates local token scores from recent attention patterns and aggregates neighboring-layer scores to derive a cross-layer consensus signal. This signal highlights shared contextual anchors, while the residual between local and consensus scores is used to identify diversity tokens carrying layer-specific information. By preserving both consensus and diversity tokens, ConsistKV retains globally shared context as well as layer-dependent information. Beyond token selection, ConsistKV estimates each layer’s cache demand from cross-layer disagreement and score dispersion, and allocates more cache capacity to layers with higher preservation needs. In this way, ConsistKV jointly determines which tokens to retain and how the total cache budget should be distributed across layers.

Our contributions are summarized as follows:

- We observe that important KV tokens exhibit structured cross-layer patterns, motivating ConsistKV to preserve both shared contextual anchors and layer-specific information.
- We propose ConsistKV, a KV cache compression method guided by cross-layer consistency. It preserves shared contextual anchors through neighboring-layer consensus, retains layer specific tokens through residual local importance, and adaptively allocates cache budgets according to layer demand.
- We conduct extensive experiments on LongBench across multiple backbone LLMs and cache retention ratios. ConsistKV achieves leading average performance among strong KV cache compression baselines under different cache budgets.

2 Related Work

KV Cache Compression Strategies. KV cache compression has become an important direction for efficient long-context LLM inference, as the memory footprint of cached keys and values grows linearly with sequence length. Early methods mainly rely on simple retention heuristics, such as preserving attention sinks and recent tokens for stable streaming generation (Xiao et al., 2024; Han et al., 2024). Subsequent studies improve eviction decisions by estimating token importance from attention statistics. H₂O (Zhang et al., 2023) identifies heavy-hitter tokens from accumulated attention scores, while SnapKV (Li et al., 2024) selects important KV positions from an observation window before generation. In addition, Quest (Tang et al., 2024) introduces query-aware sparsity by selecting critical KV pages conditioned on the current query. Although these methods reduce cache memory, they typically perform token selection independently within each layer and often assume uniform or manually designed cache budgets across layers.

Recent studies further exploit the layer structure and contextual organization of KV cache. PyramidInfer and PyramidKV (Yang et al., 2024; Cai et al., 2024) allocate different cache budgets to different layers according to their compression patterns, while CAKE (Qin et al., 2025) estimates cache preferences from spatial and temporal attention dynamics to adaptively distribute budgets. Orthogonally, ChunkKV (Liu et al., 2025) preserves semantic chunks instead of isolated tokens to reduce context fragmentation. However, these methods mainly focus on budget allocation or semantic grouping, leaving the shared and distinct importance of tokens across layers underexplored. In contrast, our method explicitly models both consensus and diversity in token importance across layers, enabling a unified compression strategy for token preservation and adaptive budget allocation.

3 Methodology

3.1 Problem Formulation

Given a decoder-only Transformer with L layers and an input sequence $\mathbf{x} = \{x_i\}_{i=1}^N$, the prefilling stage produces a full KV cache for each layer:

$$\mathcal{C}_\ell = \{(\mathbf{k}_{\ell,i}, \mathbf{v}_{\ell,i})\}_{i=1}^N, \quad (1)$$

where $\mathbf{k}_{\ell,i}$ and $\mathbf{v}_{\ell,i}$ denote the key and value states of the i -th token at layer ℓ . During autoregressive

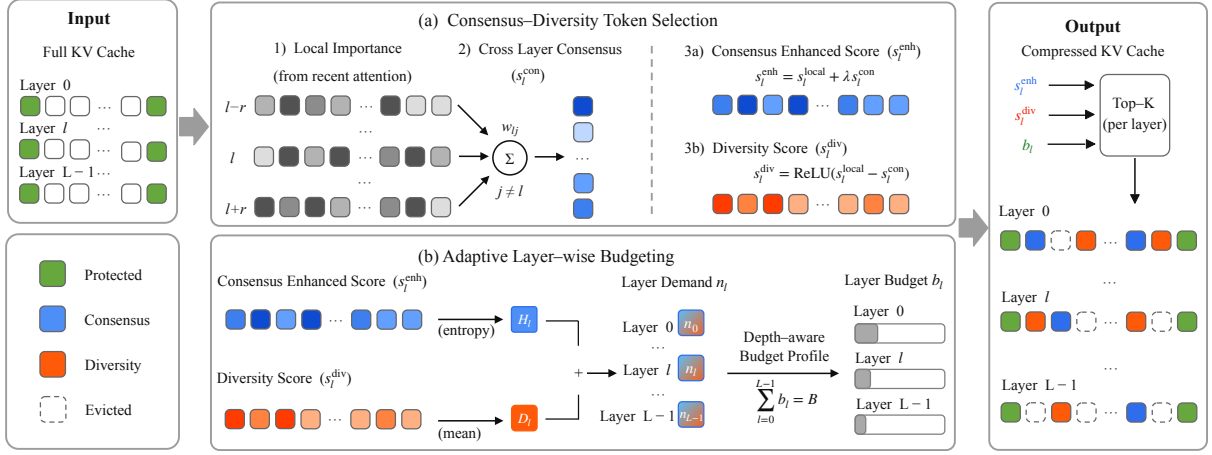


Figure 2: Overview of ConsistKV. (a) Consensus and diversity aware token preservation identifies both cross layer shared tokens and layer specific tokens. (b) Adaptive cache budget allocation assigns layer budgets according to cache demand estimated from disagreement and entropy.

decoding, each newly generated query attends to cached KV pairs from previous tokens. As a result, the KV cache memory grows linearly with the context length and the number of layers, while the decoding attention cost also increases with the retained cache length.

KV cache compression aims to replace the full cache $\mathcal{C}_\ell$ with a smaller cache $\hat{\mathcal{C}}_\ell$ by retaining only a subset of token positions. Specifically, for each layer ℓ , a compression method selects an index set $\mathcal{S}_\ell \subseteq \{1, \dots, N\}$ and keeps

$$\hat{\mathcal{C}}_\ell = \{(\mathbf{k}_{\ell,i}, \mathbf{v}_{\ell,i}) \mid i \in \mathcal{S}_\ell\}. \quad (2)$$

The number of retained positions $|\mathcal{S}_\ell| = b_\ell$ denotes the cache budget assigned to layer ℓ under a global cache budget B .

The objective of KV cache compression is to reduce memory usage and decoding attention cost while preserving the generation quality of the original full-cache model. Therefore, the central problem is to jointly determine which token positions should be retained in each layer and how the global cache budget should be allocated across layers.

3.2 Overview of ConsistKV

Given the full KV cache produced during prefilling, ConsistKV compresses it by jointly determining which token positions should be retained in each layer and how the cache budget should be allocated across layers. Unlike prior methods that estimate token importance independently within each layer, ConsistKV exploits the cross layer consistency of token importance. The intuition is that tokens consistently considered important by neighboring lay-

ers are likely to carry shared contextual information, while tokens salient only in a specific layer may encode layer specific information. Therefore, effective KV cache compression should preserve both cross layer shared information and layer specific information.

ConsistKV contains two main components. First, it performs consensus and diversity aware token preservation. For each layer, ConsistKV estimates local token importance from recent attention, aggregates neighboring layer importance distributions to obtain a cross layer consensus signal, and derives both consensus enhanced scores and diversity scores. Consensus enhanced scores prioritize tokens that are jointly important across neighboring layers, while diversity scores preserve tokens whose importance is specific to the current layer. Second, ConsistKV performs adaptive layer budget allocation by estimating each layer’s cache demand from disagreement and entropy, assigning larger budgets to layers with stronger layer specific information or more dispersed importance distributions. After the layer budgets are determined, ConsistKV constructs the compressed cache by retaining sink tokens, consensus tokens, diversity tokens, and recent tokens.

3.3 Consensus and Diversity Aware Token Preservation

Given a layer budget b_ℓ , ConsistKV determines which cached token positions should be preserved at layer ℓ . A straightforward strategy is to select tokens solely according to their layer local importance. However, such an independent selection ig-

nores the fact that token importance is not isolated across Transformer layers. In practice, some tokens remain consistently important in neighboring layers, indicating shared contextual information, whereas other tokens become salient only in particular layers and may reflect layer specific transformations. This motivates ConsistKV to decompose token importance into two complementary parts: a consensus component that captures cross layer shared importance, and a diversity component that captures layer specific residual importance. The retained cache is then constructed by preserving both consensus tokens and diversity tokens under the given layer budget.

For each layer ℓ , ConsistKV first estimates a local importance score $s_\ell^{local}(i)$ for each token position i based on the attention it receives from recent queries. To model cross layer consistency, we consider a neighboring layer set

$$\mathcal{M}(\ell) = \{j : 0 < |j - \ell| \leq r\}, \quad (3)$$

where r denotes the neighborhood range. We then aggregate the local importance distributions from neighboring layers to obtain a cross layer consensus score:

$$s_\ell^{con}(i) = \sum_{j \in \mathcal{M}(\ell)} w_{\ell j} s_j^{local}(i), \quad (4)$$

where $w_{\ell j}$ is a deterministic, input-dependent similarity weight between the token importance distributions of layers ℓ and j . In this way, layers with more similar importance patterns contribute more to the consensus signal.

Based on the consensus score, ConsistKV decomposes token importance into two complementary components. The first component is the consensus enhanced score:

$$s_\ell^{enh}(i) = s_\ell^{local}(i) + \lambda s_\ell^{con}(i), \quad (5)$$

where λ controls the strength of the cross layer consensus. This score prioritizes tokens that are important not only in the current layer but also in its neighboring layers. The second component is the diversity score:

$$s_\ell^{div}(i) = \text{ReLU}(s_\ell^{local}(i) - s_\ell^{con}(i)). \quad (6)$$

The diversity score captures the residual importance that cannot be explained by the neighboring layer consensus. It encourages the preservation of layer specific tokens and prevents the compressed

Algorithm 1 ConsistKV

Input: Full KV cache $\{C_\ell\}_{\ell=1}^L$, attention scores $\{A_\ell\}_{\ell=1}^L$, global budget B , neighborhood range r

Output: Compressed KV cache $\{\hat{C}_\ell\}_{\ell=1}^L$

Consensus and Diversity Scoring:

```

1: for  $\ell = 1$  to  $L$  do
2:    $s_\ell^{loc} \leftarrow \text{LocalScore}(A_\ell)$ 
3:    $\mathcal{M}(\ell) \leftarrow \{j : 0 < |j - \ell| \leq r\}$ 
4:    $s_\ell^{con} \leftarrow \sum_{j \in \mathcal{M}(\ell)} w_{\ell j} s_j^{loc}$ 
5:    $s_\ell^{enh} \leftarrow s_\ell^{loc} + \lambda s_\ell^{con}$ 
6:    $s_\ell^{div} \leftarrow [s_\ell^{loc} - s_\ell^{con}]_+$ 
7:    $n_\ell \leftarrow \text{Demand}(s_\ell^{div}, s_\ell^{enh})$ 
8: end for
Adaptive Budget Allocation:
9:  $\gamma \leftarrow \text{clip}(\exp[-\eta \Delta(\{n_\ell\}_{\ell=1}^L)], \gamma_{\min}, \gamma_{\max})$ 
10:  $\{b_\ell\}_{\ell=1}^L \leftarrow \text{BudgetProfile}(B, \gamma)$ 
Cache Retention:
11: for  $\ell = 1$  to  $L$  do
12:    $\Omega_\ell \leftarrow \{1, \dots, N\} \setminus (\mathcal{S}_\ell^{sink} \cup \mathcal{S}_\ell^{recent})$ 
13:    $\bar{b}_\ell \leftarrow b_\ell - |\mathcal{S}_\ell^{sink}| - |\mathcal{S}_\ell^{recent}|$ 
14:    $b_\ell^{div} \leftarrow \lfloor \rho_\ell \bar{b}_\ell \rfloor$ ,  $b_\ell^{con} \leftarrow \bar{b}_\ell - b_\ell^{div}$ 
15:    $\mathcal{S}_\ell^{con} \leftarrow \text{TopK}_{\Omega_\ell}(s_\ell^{enh}, b_\ell^{con})$ 
16:    $\mathcal{S}_\ell^{div} \leftarrow \text{TopK}_{\Omega_\ell \setminus \mathcal{S}_\ell^{con}}(s_\ell^{div}, b_\ell^{div})$ 
17:    $\mathcal{S}_\ell \leftarrow \mathcal{S}_\ell^{sink} \cup \mathcal{S}_\ell^{con} \cup \mathcal{S}_\ell^{div} \cup \mathcal{S}_\ell^{recent}$ 
18:    $\hat{C}_\ell \leftarrow C_\ell[\mathcal{S}_\ell]$ 
19: end for
20: return  $\{\hat{C}_\ell\}_{\ell=1}^L$ 

```

cache from over relying on only cross layer shared information.

Finally, ConsistKV constructs the retained token set for each layer by combining protected tokens, consensus tokens, and diversity tokens. Specifically, sink tokens and recent tokens are always preserved, denoted as $\mathcal{S}_\ell^{sink}$ and $\mathcal{S}_\ell^{recent}$, respectively. The remaining budget is

$$\bar{b}_\ell = b_\ell - |\mathcal{S}_\ell^{sink}| - |\mathcal{S}_\ell^{recent}|. \quad (7)$$

A fraction ρ_ℓ of the remaining budget is assigned to diversity tokens, and the rest is assigned to consensus tokens:

$$\begin{aligned} \mathcal{S}_\ell = & \mathcal{S}_\ell^{sink} \cup \text{TopK}(s_\ell^{enh}, b_\ell^{con}) \\ & \cup \text{TopK}(s_\ell^{div}, b_\ell^{div}) \cup \mathcal{S}_\ell^{recent}. \end{aligned} \quad (8)$$

Diversity tokens are selected after excluding the positions already chosen as consensus tokens. This design preserves stable attention anchors, recent contextual information, cross layer shared information, and layer specific information within the given layer budget.

3.4 Adaptive Cache Budget Allocation

After establishing the token preservation criterion under a fixed layer budget, we further determine how the global cache budget should be distributed

across layers. A fixed allocation neglects the heterogeneous compression sensitivity of different layers, since some layers contain stronger layer specific residual information while others exhibit more dispersed token importance. ConsistKV therefore derives a layer cache demand score from the consensus and diversity decomposition, and uses it to guide adaptive budget allocation.

We measure the cache demand of each layer using two complementary signals. The first is layer disagreement, which quantifies the amount of layer specific information. Given the diversity score s_ℓ^{div} , we define

$$D_\ell = \frac{1}{|\Omega_\ell|} \sum_{i \in \Omega_\ell} s_\ell^{div}(i), \quad (9)$$

where Ω_ℓ denotes the non protected candidate token positions at layer ℓ . A larger D_ℓ indicates that the layer contains more information not explained by cross layer consensus. The second signal is score entropy, which measures the dispersion of token importance. We first normalize the consensus enhanced scores as

$$p_{\ell,i} = \frac{s_\ell^{enh}(i)}{\sum_{j \in \Omega_\ell} s_\ell^{enh}(j)}, \quad (10)$$

and compute the normalized entropy

$$H_\ell = -\frac{1}{\log |\Omega_\ell|} \sum_{i \in \Omega_\ell} p_{\ell,i} \log p_{\ell,i}. \quad (11)$$

Higher entropy suggests that important information is distributed over more tokens, making the layer more sensitive to aggressive compression.

The two signals are combined into a unified budget need score:

$$n_\ell = \lambda_D \hat{D}_\ell + \lambda_H \hat{H}_\ell, \quad (12)$$

where $\hat{D}_\ell$ and $\hat{H}_\ell$ are normalized across layers, and λ_D and λ_H control their relative contributions. Thus, n_ℓ reflects the relative cache demand of layer ℓ , considering both layer specific residual information and the dispersion of token importance.

Given the global cache budget B , ConsistKV converts the layer need scores into layer budgets through a demand based budget profile. Let $z_\ell \in [-1, 1]$ denote the normalized layer depth. We compute a depth aware demand pressure:

$$\Delta = \frac{1}{L} \sum_{\ell=1}^L \left(\frac{n_\ell}{\frac{1}{L} \sum_{j=1}^L n_j} - 1 \right) z_\ell. \quad (13)$$

The pressure term Δ reflects whether higher cache demand is concentrated more in shallow or deep layers. ConsistKV then adjusts the budget profile using

$$\gamma = \text{clip}(\exp(-\eta\Delta), \gamma_{\min}, \gamma_{\max}), \quad (14)$$

where η controls the adaptation strength and $\gamma_{\min}, \gamma_{\max}$ bound the profile adjustment. The final layer budgets are obtained as

$$\{b_\ell\}_{\ell=1}^L = \text{BudgetProfile}(B, \gamma), \quad (15)$$

where the resulting budgets are discretized into integer token counts with residual adjustment to satisfy the prescribed global cache budget. This structured allocation enables input dependent redistribution of cache capacity while maintaining a regularized layer budget profile.

4 Experiment

4.1 Experimental Settings

Backbone Model. We evaluate ConsistKV on three widely used open-source instruction-tuned LLMs, including Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), and Gemma (Team et al., 2024). These models span different model families and parameter scales, allowing us to evaluate the robustness and generality of ConsistKV across diverse Transformer-based architectures.

Datasets. We conduct experiments on Long-Bench (Bai et al., 2024), a standard benchmark for evaluating long-context understanding. Following prior KV cache compression studies, we evaluate the model across multiple task categories, including single-document question answering, multi-document question answering, summarization, few-shot learning, synthetic tasks, and code completion. These tasks cover both natural language understanding and code-oriented long-context scenarios.

Baselines. We use Full Cache as the reference setting, where all KV states are preserved without eviction. For KV cache compression baselines, StreamingLLM (Xiao et al., 2024) retains attention sinks and recent tokens, H₂O (Zhang et al., 2023) selects tokens according to accumulated attention scores, SnapKV (Li et al., 2024) identifies important tokens from a recent observation window, and PyramidKV (Cai et al., 2024) assigns

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning		Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQ	Musique	GovReport	QMSum	MultiNews	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Llama-3.1-8B-Instruct, Full Cache																
Full Cache	30.72	47.06	55.28	59.51	51.83	32.66	35.25	24.94	27.05	91.71	40.92	10.70	100.00	52.05	44.49	46.94
Llama-3.1-8B-Instruct, KV Cache Budget = 10%																
StreamingLLM	21.67	18.44	23.03	37.24	22.28	14.06	25.03	19.13	20.05	90.70	35.31	4.00	16.00	<u>50.08</u>	49.92	29.80
H ₂ O	15.72	28.97	20.19	32.81	28.04	10.39	27.92	21.25	23.56	89.35	34.80	10.42	54.00	50.17	46.08	32.91
SnapKV	24.57	20.58	23.21	<u>44.32</u>	23.71	20.52	<u>25.15</u>	20.04	20.19	91.49	40.36	<u>6.00</u>	54.00	48.53	45.78	33.90
PyramidKV	22.82	<u>20.66</u>	<u>23.49</u>	45.98	24.69	19.20	24.98	20.62	20.14	<u>91.66</u>	39.45	<u>6.00</u>	<u>54.50</u>	48.32	46.48	<u>33.93</u>
ConsistKV	<u>23.59</u>	19.99	23.51	44.11	<u>26.53</u>	<u>19.43</u>	24.59	<u>20.67</u>	<u>20.22</u>	92.16	<u>39.46</u>	5.00	62.00	48.82	<u>47.00</u>	34.36
Llama-3.1-8B-Instruct, KV Cache Budget = 20%																
StreamingLLM	22.18	23.09	24.92	41.44	27.23	18.16	<u>28.08</u>	20.14	<u>22.68</u>	<u>91.71</u>	34.94	6.00	28.50	49.73	48.20	32.47
H ₂ O	20.36	33.74	27.47	39.07	35.45	16.34	30.43	22.93	25.56	89.83	36.91	9.54	76.50	50.78	<u>47.81</u>	37.51
SnapKV	27.72	26.06	29.68	54.34	38.04	25.22	27.75	21.40	22.70	91.81	<u>40.67</u>	8.10	<u>86.50</u>	51.53	44.22	<u>39.73</u>
PyramidKV	<u>27.65</u>	<u>26.23</u>	<u>28.72</u>	<u>54.14</u>	40.01	22.41	27.51	21.95	<u>22.73</u>	91.56	40.82	<u>9.55</u>	86.00	<u>51.06</u>	44.44	39.65
ConsistKV	26.18	25.13	28.36	53.65	<u>39.17</u>	<u>24.93</u>	27.08	<u>22.10</u>	22.58	91.46	40.55	12.05	91.50	50.53	44.34	39.97
Llama-3.1-8B-Instruct, KV Cache Budget = 30%																
StreamingLLM	24.23	28.29	26.57	43.67	30.92	21.10	29.32	21.02	<u>24.23</u>	91.49	36.79	6.25	36.50	48.38	<u>46.92</u>	34.38
H ₂ O	20.20	37.40	34.39	45.50	42.38	19.83	31.69	<u>23.26</u>	25.67	90.99	37.92	9.20	84.00	50.61	48.10	40.08
SnapKV	<u>27.20</u>	31.28	<u>38.32</u>	54.52	46.37	27.62	29.76	22.75	23.94	91.40	42.01	9.55	93.00	50.95	44.22	<u>42.20</u>
PyramidKV	27.61	31.61	37.40	<u>54.70</u>	44.10	<u>26.03</u>	28.47	22.87	23.74	91.56	<u>41.53</u>	10.50	<u>94.50</u>	<u>50.75</u>	43.99	41.96
ConsistKV	26.11	<u>31.99</u>	39.54	55.82	<u>44.28</u>	25.94	28.74	23.27	23.53	91.43	41.51	<u>10.38</u>	98.50	49.99	45.50	42.41
Mistral-7B-Instruct-v0.3, KV Cache Budget = 10%																
StreamingLLM	<u>18.67</u>	12.98	23.06	31.62	20.43	13.65	<u>25.22</u>	19.74	18.65	52.92	<u>32.90</u>	4.00	14.50	48.14	50.50	25.80
H ₂ O	9.06	26.99	24.98	20.85	23.42	10.68	28.31	21.52	24.25	85.45	39.29	<u>4.72</u>	47.50	45.23	51.71	30.93
SnapKV	18.52	13.27	30.58	31.93	22.77	18.18	25.10	20.25	20.47	87.73	28.21	3.82	<u>54.00</u>	<u>49.73</u>	<u>51.77</u>	<u>31.76</u>
PyramidKV	16.39	13.29	<u>29.96</u>	<u>33.73</u>	<u>23.23</u>	<u>16.37</u>	24.75	<u>20.57</u>	<u>20.53</u>	<u>88.01</u>	28.87	5.01	51.50	49.93	51.92	31.60
ConsistKV	18.91	<u>14.78</u>	27.70	35.44	22.81	15.93	24.65	20.31	20.20	88.29	32.14	4.53	55.25	49.06	51.92	32.13
Gemma-3-4B-It, KV Cache Budget = 10%																
StreamingLLM	16.90	21.86	20.76	31.16	23.39	11.12	21.56	17.19	17.84	85.47	34.37	2.00	8.00	51.73	50.86	27.61
H ₂ O	15.27	25.30	20.30	32.04	25.03	<u>12.83</u>	21.92	18.03	17.62	85.64	35.04	2.00	33.50	52.31	48.85	29.71
SnapKV	15.08	22.06	20.71	37.68	<u>28.93</u>	15.79	22.67	18.69	18.41	86.52	35.93	<u>5.00</u>	40.00	53.29	49.82	<u>31.37</u>
PyramidKV	13.94	<u>22.07</u>	<u>20.91</u>	36.61	28.52	11.80	<u>22.46</u>	18.19	18.29	86.12	36.36	<u>5.00</u>	41.50	<u>53.27</u>	<u>50.19</u>	31.02
ConsistKV	<u>16.07</u>	21.57	21.53	<u>36.77</u>	32.15	12.33	22.13	<u>18.44</u>	<u>18.33</u>	<u>86.38</u>	<u>36.32</u>	6.00	45.50	52.65	49.85	31.73

Table 1: Performance comparison over LongBench datasets. The best result is highlighted in bold, and the second best is underlined.

different cache budgets to different layers based on pyramidal information flow. All compression methods are evaluated under the same total cache budget for fair comparison.

Implementation Details. All experiments are implemented with Hugging Face Transformers and PyTorch, and are conducted on NVIDIA A100 80GB GPUs. We follow the standard LongBench evaluation protocol and use the task-specific decoding settings for each dataset. All KV compression methods are evaluated under cache retention ratios of 10%, 20%, and 30% of the full KV cache. Uniform methods evenly distribute the cache budget across layers, while layer-aware methods such as PyramidKV and ConsistKV keep the same total budget but allow different layers to receive different cache sizes. All baselines use their official or commonly adopted hyperparameter settings.

4.2 Main Results on LongBench

We evaluate ConsistKV on LongBench, which covers diverse long-context tasks including document QA, summarization, few-shot learning, synthetic

retrieval, and code completion. As shown in Table 1, ConsistKV achieves the best average performance among all compression methods across the reported models and cache budgets. On Llama-3.1-8B-Instruct, ConsistKV obtains average scores of 34.36, 39.97, and 42.41 under 10%, 20%, and 30% KV cache budgets, respectively. Under the most constrained 10% budget, it improves the average score over PyramidKV and SnapKV from 33.93 and 33.90 to 34.36. These results indicate that cross-layer consistency provides more reliable guidance for token preservation when the available KV cache is highly limited.

The results on Mistral-7B-Instruct-v0.3 and Gemma-3-4B-It further show the robustness of ConsistKV across different backbone models. Under the 10% KV cache budget, ConsistKV achieves average scores of 32.13 and 31.73 on the two models, outperforming the strongest baselines with scores of 31.76 and 31.37, respectively. These consistent improvements indicate that ConsistKV is not specific to Llama-3.1-8B-Instruct, but can provide effective token preservation across different

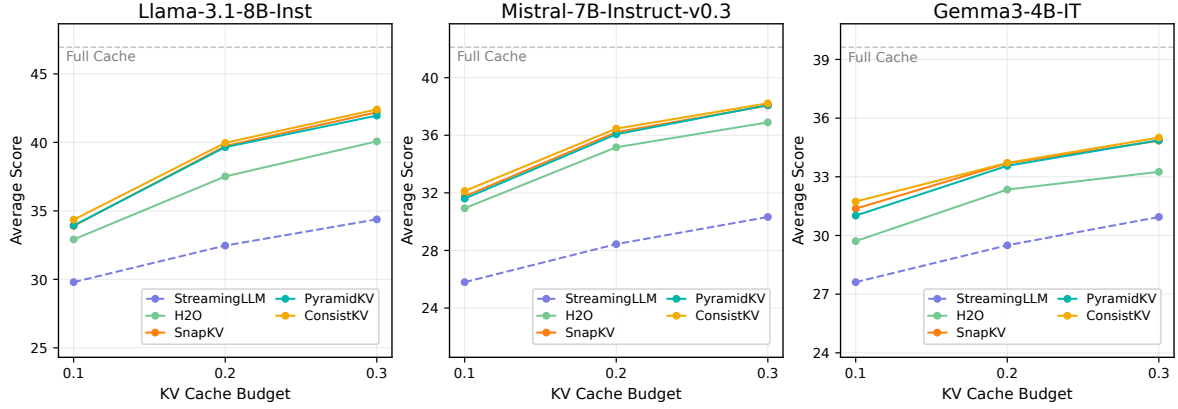


Figure 3: Average performance on LongBench under different KV cache budgets.

model families under tight cache budgets.

At the task level, ConsistKV shows clear advantages on synthetic retrieval tasks. For example, on Llama-3.1-8B-Instruct, ConsistKV achieves the best Passage Retrieval score under all three cache budgets, reaching 62.00, 91.50, and 98.50 with 10%, 20%, and 30% KV cache budgets, respectively. This indicates that cross-layer consensus is particularly helpful for preserving tokens that serve as important retrieval anchors in long contexts. Meanwhile, ConsistKV also maintains competitive performance across QA, summarization, and code tasks, leading to the best overall average score in all reported settings. Although some baselines achieve higher scores on individual sub-tasks, their gains are less consistent, whereas ConsistKV provides a more balanced preservation of long-context information.

Figure 3 further summarizes the average LongBench score under different KV cache budgets. As the retained KV cache ratio increases from 10% to 30%, all compression methods generally achieve better performance, indicating that a larger cache budget can preserve more contextual information. Across the three backbone models, ConsistKV consistently achieves the highest or competitive average scores, with more visible advantages under tighter cache budgets. This trend suggests that, when only a small fraction of KV cache can be retained, the proposed consistency-aware selection strategy provides more reliable token preservation than purely local or fixed allocation based methods.

4.3 Ablation Study

We conduct ablation studies on LongBench under the 10% KV cache budget to evaluate the key components of ConsistKV. Specifically, we compare

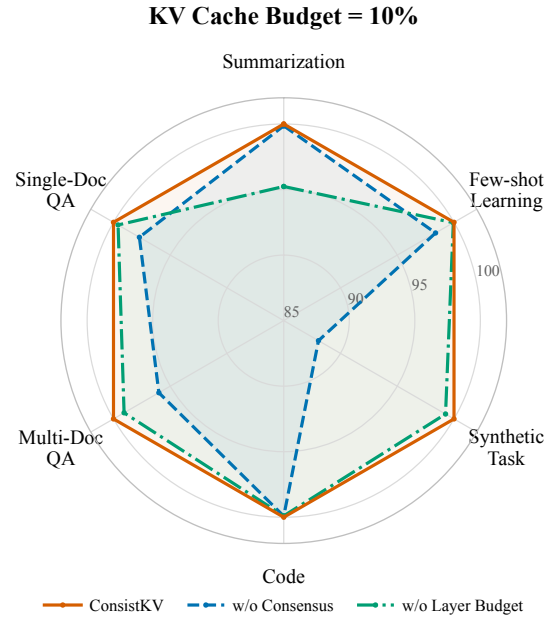


Figure 4: Ablation study under the 10% KV cache budget. The radar chart reports the relative performance of ConsistKV and its ablated variants across different LongBench task categories.

the full method with two variants: *w/o Consensus* and *w/o Layer Budget*. The *w/o Consensus* variant removes the cross-layer consensus mechanism and selects tokens only according to layer-local importance. The *w/o Layer Budget* variant disables adaptive layer budget allocation while keeping the same total cache budget. These two variants allow us to examine whether the gains of ConsistKV come from consensus-aware token selection and adaptive cache allocation.

As shown in Figure 4, both variants perform worse than the full ConsistKV across multiple task categories. Removing cross-layer consensus

Method	Input	Output	Total Latency (s) ↓	Decode Throughput (T/s) ↑
Full Cache	4096	1024	25.98	40.56
ConsistKV	4096	1024	24.63 (5.2%)	43.16 (6.4%)
Full Cache	4096	4096	93.23	44.36
ConsistKV	4096	4096	90.76 (2.6%)	45.85 (3.4%)
Full Cache	8192	1024	27.88	37.20
ConsistKV	8192	1024	25.29 (9.3%)	41.37 (11.2%)
Full Cache	8192	4096	107.95	38.07
ConsistKV	8192	4096	100.88 (6.6%)	40.78 (7.1%)

Table 2: Efficiency comparison between Full Cache and ConsistKV under different input and output lengths. Percentages in parentheses indicate latency reduction or throughput improvement over Full Cache.

leads to noticeable degradation, especially on multi-document QA and synthetic tasks, suggesting that consensus tokens act as important contextual anchors shared across layers. Removing adaptive layer budgeting also reduces performance on several tasks, indicating that uniformly or fixedly assigning cache budgets cannot fully satisfy the different preservation needs of layers. Overall, these results show that the two designs are complementary: consensus-aware selection improves the quality of retained tokens, while adaptive layer budgeting improves the utilization of limited cache capacity.

Beyond the component ablations in this section, we further provide hyperparameter sensitivity analysis in Appendix B. The results examine the effects of budget adaptation strength η , diversity ratio ρ , and consensus weight λ , and show that ConsistKV remains stable under reasonable hyperparameter variations.

4.4 Efficiency Evaluation

We further evaluate the inference efficiency of ConsistKV under different input and output lengths, using Full Cache as the reference. As shown in Table 2, ConsistKV consistently reduces total latency and improves decoding throughput across all tested settings. For input length 4096, ConsistKV reduces total latency by 5.2% when generating 1024 tokens and by 2.6% when generating 4096 tokens, while improving decoding throughput by 6.4% and 3.4%, respectively. For input length 8192, the gains become more pronounced: ConsistKV achieves 9.3% latency reduction and 11.2% throughput improvement with 1024 output tokens, and still obtains 6.6% latency reduction and 7.1% throughput improvement with 4096 output tokens.

These results show that ConsistKV brings prac-

Method	10%	20%	30%
StreamingLLM	7.0	19.0	35.0
SnapKV	19.0	39.0	62.0
PyramidKV	18.0	34.0	73.0
ConsistKV	20.0	56.0	80.0
Gain	+1.0	+17.0	+7.0

Table 3: Performance on the 32K RULER niah_multikey_1 task under different KV retention ratios. Gain denotes the improvement of ConsistKV over the strongest competing baseline at each retention ratio.

tical efficiency benefits in addition to preserving task performance. The improvement is larger under longer input contexts, where the cost of attending to a full KV cache becomes more significant during decoding. By retaining a compact subset of informative KV states, ConsistKV reduces the effective cache size accessed during generation, thereby lowering total latency and increasing decoding throughput. This indicates that the proposed cache compression strategy is useful not only for memory reduction, but also for accelerating long-context inference under different generation lengths.

Beyond efficiency gains, ConsistKV is effective under tight cache budgets. When only a small fraction of KV states can be retained, purely layer-local selection may miss tokens serving as shared contextual anchors. Cross-layer consensus provides a more stable criterion for preserving such tokens, while the diversity score retains layer-dependent information. Meanwhile, adaptive layer budgeting prioritizes layers with higher disagreement and more dispersed importance, improving utilization of limited budgets.

4.5 Evaluation on RULER

We further evaluate ConsistKV under long context settings on the 32K niah_multikey_1 task from RULER, using the same configuration as in the LongBench experiments.

As shown in Table 3, ConsistKV achieves the best performance across all three KV retention ratios. Specifically, it outperforms the strongest competing baseline by 1.0, 17.0, and 7.0 points at the 10%, 20%, and 30% retention ratios, respectively. The improvements are particularly substantial at the 20% and 30% retention ratios, where ConsistKV maintains a clear advantage over all competing methods. Moreover, the results show that its advantage persists at a 32K context length,

suggesting that consensus across neighboring layers remains useful as the context grows. Overall, these results provide further evidence for the effectiveness of ConsistKV under long context settings.

5 Conclusion

In this paper, we proposed ConsistKV, an adaptive KV cache compression method for long-context LLM inference. ConsistKV exploits consensus in token importance across layers to preserve shared contextual anchors, while retaining tokens salient to individual layers through residual local importance. It further estimates layer cache demand from disagreement between layers and importance dispersion, enabling adaptive budget allocation under a fixed global cache budget. Experiments on LongBench across multiple LLMs and cache budgets show that ConsistKV achieves the highest average performance among the compared methods under identical cache budgets. These results demonstrate the effectiveness of modeling shared and specialized KV importance across layers.

Limitations

Despite its effectiveness, ConsistKV has several limitations. First, our evaluation is mainly limited to LongBench and several open-source instruction-tuned LLMs, which may not fully reflect scenarios with specialized domains or different generation settings. Since ConsistKV relies on attention-derived importance, it may also miss fine-grained information that receives weak attention but remains useful under very tight cache budgets. Second, ConsistKV assumes useful consensus patterns among neighboring Transformer layers. Although this assumption is supported by our empirical analysis on dense Transformer models, its applicability to architectures with different layer behaviors, such as Mixture-of-Experts models, state-space models, or highly heterogeneous attention patterns, remains less clear.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62572336, 62272334).

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham

Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multi-task benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008.

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

Ehsan Kamalloo, Nouha Dziri, Charles Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. In *Proceedings of the 61st annual meeting of the Association for Computational Linguistics (volume 1: long papers)*, pages 5591–5606.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025.

- Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 38:28728–28778.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhao Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.
- Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. **CAKE: cascading and adaptive KV cache eviction with layer preferences**. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. 2023. Zeroscrolls: A zero-shot benchmark for long text understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7977–7989.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. **QUEST: query-aware sparsity for efficient long-context LLM inference**. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 47901–47911. PMLR / OpenReview.net.
- Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, and 89 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*.
- Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3258–3270.
- Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. 2024. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710.

A Implementation Details

All experiments are implemented with Hugging Face Transformers and PyTorch, and are conducted on NVIDIA A100 80GB GPUs. We follow the standard LongBench evaluation protocol and use the task-specific decoding settings for each dataset. For all KV cache compression methods, we evaluate cache retention ratios of 10%, 20%, and 30% of the full KV cache. All methods are compared under the same total KV cache budget. For layer-aware methods, the total budget is kept identical to other methods, while different layers are allowed to receive different cache sizes.

In our implementation, the compression ratio is specified as the fraction of KV states to be removed. Thus, the 10%, 20%, and 30% cache retention ratios reported in the paper correspond to implementation compression ratios of 90%, 80%, and 70%, respectively. For all reported results, sink tokens and recent tokens are treated as protected tokens and are always retained before selecting the remaining tokens according to consensus-enhanced and diversity scores. Table 4 summarizes the default hyperparameter settings of ConsistKV. Unless otherwise specified, all ablation studies and hyperparameter analyses use these default values.

For local token importance estimation, ConsistKV uses a recent attention window of 64 queries and applies average pooling with a kernel size of 5 to smooth token-level importance scores. The recent window is set to the same value as the local attention window. For cross-layer consensus, we use a neighboring layer range of $r = 2$. The consensus-enhanced score is computed with $\lambda = 0.5$, which balances layer-local importance and neighboring-layer consensus. The base diversity ratio is set to

Hyperparameter	Symbol	Value
Neighborhood range	r	2
Consensus weight	λ	0.5
Diversity ratio	ρ	0.1
Local attention window	w	64
Score smoothing kernel	k	5
Sink tokens	n_{sink}	4
Recent tokens	n_{recent}	64
Budget slope	β	20
Budget adaptation strength	η	0.75
Disagreement weight	λ_D	1.0
Entropy weight	λ_H	0.5

Table 4: Default hyperparameter settings of ConsistKV.

Hyperparameter	Value	Avg. Score
Budget adaptation strength η	0.25	34.33
	0.50	34.24
	0.75 (default)	34.36
Diversity ratio ρ	0.10 (default)	34.36
	0.20	34.31
	0.30	34.20
Consensus weight λ	0.30	33.91
	0.50 (default)	34.36
	0.70	33.87

Table 5: Hyperparameter sensitivity analysis under the 10% KV cache budget on Llama-3.1-8B-Instruct.

$\rho = 0.1$, meaning that 10% of the non-protected layer budget is reserved for diversity tokens by default. When adaptive diversity is enabled, the layer-wise diversity ratio is adjusted according to cross-layer disagreement and clipped to $[0.05, 0.2]$.

For layer-wise budget allocation, ConsistKV uses a PyramidKV-style budget profile as the base allocation and adaptively adjusts its shape according to the estimated layer cache demand. The base budget slope is controlled by $\beta = 20$. The budget adaptation strength is set to $\eta = 0.75$, and the adaptive beta scale is clipped to $[0.5, 2.0]$ to avoid overly flat or overly sharp budget profiles. The layer cache demand is computed from cross-layer disagreement and score entropy, with weights $\lambda_D = 1.0$ and $\lambda_H = 0.5$, respectively.

B Hyperparameter Sensitivity

We analyze the sensitivity of ConsistKV under the 10% KV cache budget on Llama-3.1-8B-Instruct. We vary one hyperparameter at a time while keeping all other settings unchanged and report the average LongBench score. The default setting is $\eta = 0.75$, $\rho = 0.1$, and $\lambda = 0.5$.

As shown in Table 5, ConsistKV remains stable

Context	Method	Lat. (s)	Tok./s	Mem. (GB)
32K	FullKV	36.77	31.34	19.16
32K	StreamingLLM	27.16	44.44	15.91
32K	SnapKV	26.98	45.01	15.91
32K	PyramidKV	33.27	35.72	15.94
32K	ConsistKV	28.63	43.56	15.94
64K	FullKV	60.49	20.86	23.22
64K	StreamingLLM	36.89	40.99	16.72
64K	SnapKV	35.67	42.78	16.72
64K	PyramidKV	34.74	44.36	16.75
64K	ConsistKV	35.70	44.17	16.74
128K	FullKV	119.41	12.37	31.34
128K	StreamingLLM	66.91	33.86	18.34
128K	SnapKV	68.65	32.46	18.34
128K	PyramidKV	66.43	34.88	18.39
128K	ConsistKV	66.54	34.86	18.39

Table 6: End-to-end efficiency comparison at context lengths from 32K to 128K under identical experimental settings.

across different hyperparameter choices. For the budget adaptation strength η , the default value of 0.75 achieves the best average score, while smaller values lead to only mild performance drops. This suggests that moderately strengthening the adaptive adjustment of the layer-wise budget profile is beneficial for matching layer-specific cache demand.

For the diversity ratio ρ , the best performance is obtained at the default value of 0.1. Increasing ρ to 0.2 or 0.3 slightly decreases the average score, indicating that reserving a small portion of the non-protected budget for diversity tokens is sufficient. Allocating too much budget to diversity tokens may reduce the capacity for consensus-enhanced tokens, which serve as shared contextual anchors across layers.

For the consensus weight λ , the default value of 0.5 achieves the best result. Both smaller and larger values lead to performance degradation, suggesting that ConsistKV benefits from a balanced combination of layer-local importance and cross-layer consensus. Overall, these results show that ConsistKV does not rely on a highly sensitive hyperparameter configuration and remains robust under reasonable hyperparameter variations.

C Further Efficiency Analysis

We further compare the end-to-end efficiency of ConsistKV with FullKV, StreamingLLM, SnapKV, and PyramidKV under identical experimental settings. Table 6 reports total latency, generation throughput, and peak GPU memory at context lengths of 32K, 64K, and 128K.

As shown in Table 6, ConsistKV has nearly iden-

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning		Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQ	Musique	GovReport	QMSum	MultiNews	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
Mistral-7B-Instruct-v0.3, Full Cache																
Full Cache	28.39	40.64	51.99	48.58	37.59	27.45	34.29	25.87	27.02	85.04	20.88	5.18	98.00	47.10	53.63	42.11
Mistral-7B-Instruct-v0.3, KV Cache Budget = 10%																
StreamingLLM	18.67	12.98	23.06	31.62	20.43	13.65	25.22	19.74	18.65	52.92	32.90	4.00	14.50	48.14	50.50	25.80
H ₂ O	9.06	26.99	24.98	20.85	23.42	10.68	28.31	21.52	24.25	85.45	39.29	4.72	47.50	45.23	51.71	30.93
SnapKV	18.52	13.27	30.58	31.93	22.77	18.18	25.10	20.25	20.47	87.73	28.21	3.82	54.00	49.73	51.77	31.76
PyramidKV	16.39	13.29	29.96	33.73	23.23	16.37	24.75	20.57	20.53	88.01	28.87	5.01	51.50	49.93	51.92	31.60
ConsistKV	18.91	14.78	27.70	35.44	22.81	15.93	24.65	20.31	20.20	88.29	32.14	4.53	55.25	49.06	51.92	32.13
Mistral-7B-Instruct-v0.3, KV Cache Budget = 20%																
StreamingLLM	19.72	17.60	24.67	37.44	24.02	14.76	28.00	21.33	21.99	59.12	27.85	3.07	27.50	49.48	50.04	28.49
H ₂ O	11.84	30.92	32.93	25.12	27.25	12.02	30.45	21.22	25.09	87.73	39.16	5.08	82.50	45.54	50.63	35.17
SnapKV	20.80	19.90	35.83	39.85	27.34	20.14	28.09	21.25	22.48	86.67	24.70	4.79	86.50	52.09	52.65	36.21
PyramidKV	20.38	19.47	34.19	40.51	28.67	17.87	26.91	21.33	22.50	87.96	26.54	3.44	86.50	51.78	52.99	36.07
ConsistKV	19.34	21.70	33.34	40.45	27.83	19.61	26.82	21.67	22.31	88.46	27.25	3.88	91.50	50.26	52.46	36.46
Mistral-7B-Instruct-v0.3, KV Cache Budget = 30%																
StreamingLLM	20.25	21.65	27.78	38.15	26.76	17.67	29.57	22.03	23.69	63.93	25.94	1.50	36.00	50.00	49.96	30.33
H ₂ O	14.71	32.12	36.63	31.18	31.70	12.62	32.18	22.47	25.80	88.98	34.26	4.57	89.50	46.04	50.68	36.90
SnapKV	22.29	25.02	40.80	43.47	29.13	21.43	29.61	22.20	23.87	87.26	22.42	5.94	93.00	51.04	53.52	38.07
PyramidKV	20.79	25.63	40.98	41.60	31.41	22.43	28.36	22.49	24.06	88.39	23.90	3.44	94.50	50.69	52.65	38.09
ConsistKV	23.14	24.44	39.19	40.91	33.01	22.08	28.47	22.53	23.52	88.79	24.44	4.22	96.00	49.86	52.68	38.22
Gemma-3-4B-It, Full Cache																
Full Cache	20.90	42.57	52.97	46.48	37.84	29.19	27.46	20.55	21.96	87.28	40.16	5.50	59.00	52.60	49.86	39.62
Gemma-3-4B-It, KV Cache Budget = 10%																
StreamingLLM	16.90	21.86	20.76	31.16	23.39	11.12	21.56	17.19	17.84	85.47	34.37	2.00	8.00	51.73	50.86	27.61
H ₂ O	15.27	25.30	20.30	32.04	25.03	12.83	21.92	18.03	17.62	85.64	35.04	2.00	33.50	52.31	48.85	29.71
SnapKV	15.08	22.06	20.71	37.68	28.93	15.79	22.67	18.69	18.41	86.52	35.93	5.00	40.00	53.29	49.82	31.37
PyramidKV	13.94	22.07	20.91	36.61	28.52	11.80	22.46	18.19	18.29	86.12	36.36	5.00	41.50	53.27	50.19	31.02
ConsistKV	16.07	21.57	21.53	36.77	32.15	12.33	22.13	18.44	18.33	86.38	36.32	6.00	45.50	52.65	49.85	31.73
Gemma-3-4B-It, KV Cache Budget = 20%																
StreamingLLM	17.52	24.58	23.77	35.58	25.47	14.18	23.05	17.69	18.67	85.52	36.37	2.50	14.50	51.83	51.25	29.50
H ₂ O	16.94	29.77	25.16	35.88	26.16	14.97	22.93	18.48	28.76	84.40	37.09	3.50	40.50	52.22	48.44	32.35
SnapKV	17.83	26.14	29.88	40.07	29.13	16.59	24.26	18.49	19.33	85.58	38.38	6.50	50.50	53.21	49.44	33.69
PyramidKV	16.28	25.88	28.46	38.24	30.15	15.82	23.75	18.39	19.35	87.38	38.83	6.50	51.00	53.87	49.50	33.56
ConsistKV	17.04	24.92	29.29	37.30	32.03	14.66	23.41	18.45	19.29	86.88	38.84	7.50	55.00	52.68	48.37	33.71
Gemma-3-4B-It, KV Cache Budget = 30%																
StreamingLLM	17.97	28.16	24.93	37.71	28.68	18.68	24.07	18.01	19.62	86.66	37.57	2.50	16.50	52.40	50.71	30.94
H ₂ O	17.70	34.74	30.26	36.79	28.62	16.04	23.18	19.27	19.45	84.34	38.12	3.75	45.50	52.86	48.19	33.25
SnapKV	18.88	29.39	34.02	42.52	31.57	18.57	24.85	19.13	20.05	85.28	39.51	4.50	52.00	53.17	49.20	34.84
PyramidKV	18.94	28.22	32.45	39.90	30.60	18.36	24.56	18.96	19.98	87.78	40.25	6.50	55.00	52.72	48.81	34.87
ConsistKV	16.34	28.21	31.32	42.96	32.41	16.16	24.46	18.58	20.03	87.78	39.73	6.00	59.50	52.58	49.02	35.05

Table 7: Performance comparison over LongBench datasets on Mistral-7B-Instruct-v0.3 and Gemma-3-4B-It. The best result is highlighted in bold, and the second best is underlined.

tical peak memory usage to the compression baselines and remains competitive in both latency and throughput across long-context settings. At 128K, its latency and throughput are essentially the same as those of PyramidKV, while ConsistKV achieves lower latency and higher throughput than SnapKV. Compared with FullKV, ConsistKV reduces total latency by 44.3% and peak memory by 41.3%. These results show that the quality gains of ConsistKV do not introduce a material system-level efficiency penalty.