# SlimKV: Joint Token-Feature KV Cache Compression with Reconstruction-Free Beacon Attention

Anonymous ACL submission


## Abstract

Long-context LLM serving is increasingly bottlenecked by KV-cache memory, especially in resource-constrained scenarios. Among existing KV-cache compression strategies, token-wise methods reduce cached states but risk information loss through eviction or condensation, while feature-wise methods reduce per-token KV dimensions but can require full-dimensional reconstruction to apply positional embedding, limiting decoding speedups. We introduce SlimKV, a question-agnostic joint token-feature KV-cache compression method. SlimKV uses low-rank-aware training to compress long contexts into beacon memory states with latent KV representations, together with layer-adaptive ranks allocation. We further uncover a positional asymmetry: removing key-side RoPE affects beacon and raw tokens differently, with much smaller degradation for beacon tokens. Exploiting this asymmetry, SlimKV trains beacon KV projections under a K-RoPE-free constraint and enables latent-space attention during decoding, mitigating reconstruction latency. On LongBench, SlimKV outperforms baselines at 16×/32× compression and remains leading at 4×/8×, where it retains over 96% of the uncompressed model's score. Needle-in-a-Haystack confirms robustness across evidence positions, and efficiency evaluation shows up to 7.34× attention speedup and 3.38× end-to-end decoding speedup over the uncompressed model at 128K length.


## 1 Introduction

Large language models (LLMs) have shown strong performance across diverse tasks, and long-context capability is crucial for document understanding, multi-turn interaction, and complex reasoning (Tworkowski et al., 2023; Liu et al., 2025). However, during long-context inference, the key-value (KV) cache grows rapidly with context length, causing substantial memory pressure and memory-access latency (Xiao et al., 2024). KV-cache overhead therefore becomes a major challenge for efficient deployment under limited memory and bandwidth (Kwon et al., 2023).

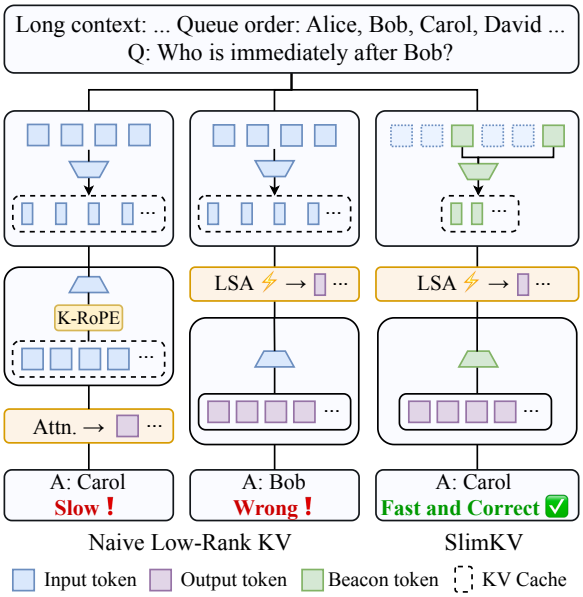


Figure 1: Comparison of low-rank KV-cache strategies. K-RoPE forces reconstruction, while its removal enables latent-space attention (LSA) but loses order information. SlimKV uses K-RoPE-free beacon tokens to retain semantics and positional cues in latent KV space.

Recent methods compress long-context KV caches mainly along the token or feature dimension. Token-wise methods reduce cached states, but hard deletion or selection can lose information under aggressive compression (Kim et al., 2026), and eviction-based methods often require full KV construction before pruning (Zhang et al., 2023; Li et al., 2024). Feature-wise methods reduce per-state dimensionality, but post-hoc low-rank compression can introduce information bottlenecks at high compression ratios (Saxena et al., 2024; Chang et al., 2025). Activation Beacon (Zhang et al., 2025) alleviates this problem by learning

beacon states that softly compress historical context during prefill. However, these states remain full-dimensional, leaving feature redundancy underexplored. Moreover, a naive combination with low-rank feature compression may introduce additional reconstruction overhead. This motivates our central question:

*Can joint token-feature compression overcome single-axis limits while avoiding the reconstruction bottleneck of low-rank decoding?*

To answer this question, we propose **SlimKV**, a joint token-feature KV-cache compression framework built upon beacon-based soft compression (Zhang et al., 2025). SlimKV trains beacon KV projections in a low-rank-aware form, allowing each beacon state to summarize historical tokens while forming compact latent KV representations. To better use a fixed feature budget, SlimKV further allocates adaptive ranks across layers according to layer-wise low-rank heterogeneity.

A remaining challenge is decoding latency: in RoPE-based backbones (Su et al., 2024), low-rank Keys must usually be reconstructed before key-side rotation and attention scoring, limiting the benefit of feature-wise compression (Saxena et al., 2024; Chang et al., 2025). Under the assumption that compressing multiple tokens into a beacon state can encode local order within its dense representation, SlimKV exploits an empirical positional asymmetry between beacon and raw-token Keys: removing key-side RoPE affects both beacon and raw-token Keys, yet beacon memory tokens exhibit much smaller performance degradation than raw tokens (Figure 2). We therefore remove K-RoPE only from beacon Keys while preserving RoPE for raw-token Keys and all Queries, enabling reconstruction-free latent-space attention.

We evaluate SlimKV on LongBench, Needle-in-a-Haystack, and long-context decoding benchmarks, covering accuracy, positional robustness, and latency.

Our contributions are threefold:

- We propose **SlimKV**, a question-agnostic joint token-feature KV-cache compression framework that learns low-rank beacon KV representations and allocates layer-adaptive ranks under a fixed feature budget.
- We identify an asymmetric dependence on key-side RoPE between beacon and raw-token Keys, and strengthen this property to enable reconstruction-free latent-space attention by removing K-RoPE only from beacon Keys.
- Experiments show that SlimKV achieves clear advantages under high compression, remains leading at lower compression ratios while retaining over 96% of the uncompressed model's LongBench score, and achieves up to 7.34× attention speedup and 3.38× end-to-end decoding speedup.

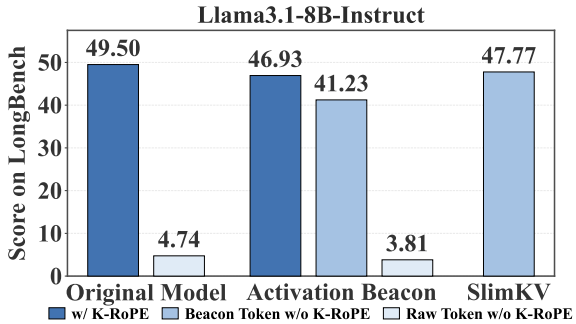


Figure 2: LongBench scores under K-RoPE ablations, showing positional asymmetry between raw and beacon tokens. SlimKV remains close to the original model without beacon K-RoPE.

## 2 Related Work

**Token-wise Compression** Token-wise KV-cache compression reduces cache size by decreasing the number of historical states used in attention. Existing methods include *prompt-level* and *KV-level* compression. *Prompt-level* methods operate on the input context: hard prompt compression (Jiang et al., 2023, 2024) removes selected tokens, while soft prompt compression (Mu et al., 2023; Ge et al., 2024; Li et al., 2025; Zhang et al., 2025) condenses long prompts into learned memory tokens. *KV-level* methods compress generated caches through KV eviction (Zhang et al., 2023; Li et al., 2024; Kim et al., 2026) or KV merging (Zhang et al., 2024; Wang et al., 2024). However, deletion-based methods may discard useful information, and question-aware variants are less suitable when future queries are unavailable. Soft compression and KV merging avoid explicit deletion but still face capacity bottlenecks under aggressive compression; KV merging also often relies on heuristic aggregation. Moreover, KV-level compression reduces decoding-time cache cost but not prefill computation for the original long prompt.

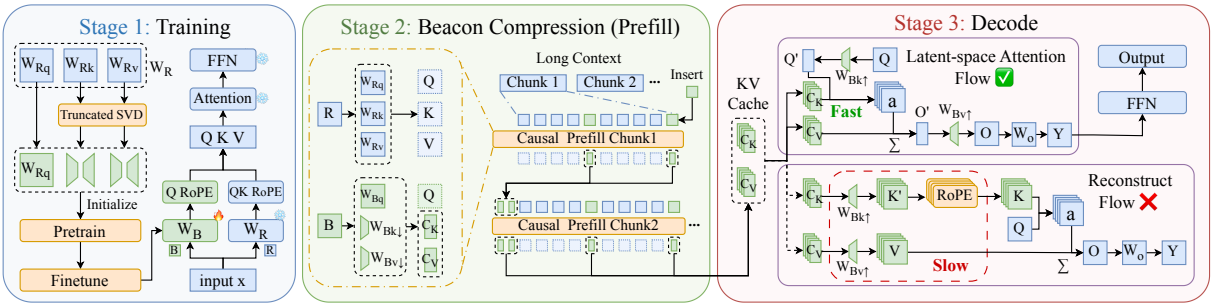


Figure 3: Overview of SlimKV. SlimKV trains low-rank beacon KV projections and caches only latent beacon KV states after prefill. During decoding, SlimKV performs latent-space attention over cached beacon memories, avoiding full-dimensional reconstruction.

**Feature-wise Compression** Feature-wise KV-cache compression reduces per-token cache size by shrinking feature representations or removing head redundancy. MQA (Shazeer, 2019) and GQA (Ainslie et al., 2023) share or group K/V heads, while latent-KV architectures such as MLA (Liu et al., 2024a) store K/V in a low-dimensional latent space with decoupled positional handling. However, MLA-style designs typically require pre-training from scratch and are less applicable to existing GQA-based backbones. Post-training methods such as Eigen Attention (Saxena et al., 2024), Palu (Chang et al., 2025), and MHA2MLA (Ji et al., 2025) retrofit existing models with low-rank latent KV structures. Nevertheless, post-hoc SVD-style decomposition may introduce information bottlenecks, and low-rank Keys often require full-space reconstruction before RoPE and attention scoring, incurring additional latency.

## 3 Joint Token-Feature Compression

### 3.1 Overview

SlimKV is a joint token-feature KV-cache compression framework built on the Activation Beacon-style progressive token compression pipeline (Zhang et al., 2025). The input is chunked with beacon tokens inserted at fixed intervals; each chunk attends to previous beacon memories, allowing beacon states to absorb contextual information. After each chunk-wise prefill, raw-token KV states are discarded and only compressed beacon memories are cached.

SlimKV further extends this pipeline to feature-wise compression by training beacon KV projections directly in their target low-rank form, rather than applying post-hoc decomposition. It caches only latent beacon Keys and Values and allocates layer-adaptive ranks under a global compression budget to match layer-wise heterogeneity. Figure 3 provides an overview.

### 3.2 Low-Rank Beacon Training

SlimKV applies low-rank parameterization only to the beacon branch, with the backbone frozen and only beacon projection parameters updated during training. Let $h_b^{(l)}$ denote a beacon hidden state at layer $l$, and let $D$ be the original KV dimension. Instead of generating full-dimensional beacon Keys and Values, SlimKV first maps beacon states into a layer-specific latent space:

$$C_K^{(l)} = h_b^{(l)} W_{K,\downarrow}^{(l)}, \quad C_V^{(l)} = h_b^{(l)} W_{V,\downarrow}^{(l)}, \tag{1}$$

where both $C_K^{(l)}$ and $C_V^{(l)}$ have dimensions lower than $D$. The corresponding full-dimensional forms for head $m$ can be written as

$$K_b^{(l)} = C_K^{(l)} U_K^{(l)\top}, \quad V_b^{(l)} = C_V^{(l)} U_V^{(l)\top}. \tag{2}$$

During inference, only the latent states $C_K^{(l)}$ and $C_V^{(l)}$ are stored in the KV cache.

Unlike post-training low-rank compression, SlimKV trains beacon KV projections under the target rank constraint from the beginning, allowing beacon memory to adapt to the target latent dimension. Training follows the standard causal language modeling objective over raw tokens. For a raw-token position $t$, the visible context is

$$\mathcal{C}_t = \{\mathcal{B}_{\text{prev}}, \mathcal{B}_{<t,\text{cur}}, \mathcal{R}_{<t,\text{cur}}\}, \tag{3}$$

where $\mathcal{B}_{\text{prev}}$ denotes cached latent beacon memory from previous chunks, $\mathcal{B}_{<t,\text{cur}}$ denotes latent beacon memory from the current-chunk prefix, and $\mathcal{R}_{<t,\text{cur}}$ denotes raw-token states from the current-chunk prefix. The training objective is

$$\mathcal{L} = -\sum_{t \in \mathcal{T}_{\text{raw}}} \log p_\theta\left(x_t \mid \mathcal{C}_t\right), \tag{4}$$

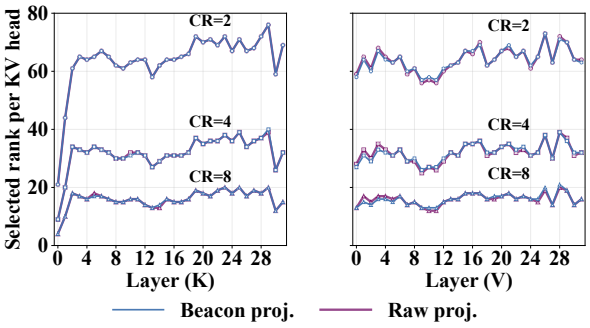


Figure 4: Layer-wise rank allocation. K/V ranks of Beacon and raw weights in Llama-3.1-8B under 2×, 4×, and 8× feature compression, selected by equal explained variance.

where $\mathcal{T}_{\text{raw}}$ denotes non-beacon positions and $\theta$ denotes model parameters. Beacon tokens are excluded from the loss.

### 3.3 Layer-Adaptive Rank Allocation

A uniform latent rank ignores the heterogeneous low-rank structure of Transformer projections. Since trained beacon projections are unavailable before training, SlimKV uses pretrained raw-token KV projection weights as proxies. Their layer-wise rank patterns are highly consistent with those of beacon projection weights, as shown in Figure 4 and Table 8. SlimKV therefore derives beacon ranks from raw-token singular-value spectra and initializes each low-rank beacon projection with the truncated SVD of its raw-token counterpart.

We use a floor-constrained equal-energy criterion. Let $\sigma_{l,1} \geq \cdots \geq \sigma_{l,D}$ denote the singular values of the pretrained raw-token KV projection at layer $l$, and define the cumulative spectral energy as

$$E_l(r) = \frac{\sum_{j=1}^{r} \sigma_{l,j}^2}{\sum_{j=1}^{D} \sigma_{l,j}^2}. \tag{5}$$

Given an energy threshold $\rho$, the initial rank is

$$r_l^0 = \min\{r \mid E_l(r) \geq \rho\}. \tag{6}$$

Pure equal-energy allocation may assign very small ranks to layers with concentrated spectra, creating training bottlenecks. We therefore impose a rank floor. Given a total rank budget $B$ and $L$ layers, where $B = LD/k$ for feature compression ratio $k$, let $\bar{r} = B/L$ be the average rank and set

$$r_{\min} = \lceil \lambda \bar{r} \rceil. \tag{7}$$

where $\lambda$ is a hyperparameter that controls the minimum rank. The final rank is

$$r_l(\rho) = \max(r_l^0, r_{\min}). \tag{8}$$

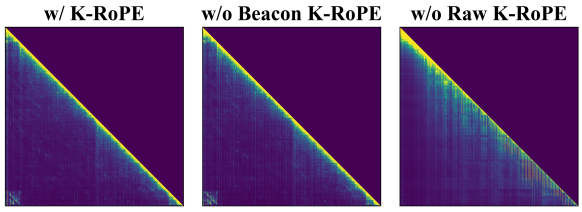


Figure 5: Attention maps under K-RoPE ablations. Beacon K-RoPE removal remains much closer to the K-RoPE-kept pattern than raw-token K-RoPE removal.

Table 1: Similarity to the w/ K-RoPE setting.

| Variant | Attn. ↑ | Block ↑ | Lag JS ↓ | Col. JS ↓ |
|---|---|---|---|---|
| Beacon removed | 0.9707 | 0.9982 | 0.0002 | 0.0030 |
| Raw removed | 0.7570 | 0.8922 | 0.0062 | 0.0349 |

We choose the largest $\rho$ that satisfies the global budget:

$$\rho^\star = \max\{\rho \mid \sum_{l=1}^{L} r_l(\rho) \leq B\}. \tag{9}$$

Any unused rank budget after integer assignment is allocated to layers with the largest marginal energy gain. This preserves balanced spectral energy, avoids under-allocation, and fixes all beacon ranks.

## 4 From Compression to Acceleration: Latent-Space Attention

The previous section reduces KV-cache size, but Key reconstruction can still limit speed. Can compressed beacon Keys enable latent-space decoding without full-Key reconstruction?

### 4.1 RoPE-aware Beacon Generation

Although cached beacon Keys are RoPE-free, beacon Queries still retain RoPE and attend to RoPE-applied raw-token Keys during beacon generation. Thus, beacon states are formed through RoPE-aware QK interactions (Su et al., 2024):

$$\alpha_{b,i} = \mathrm{softmax}_i \left( \mathrm{RoPE}(q_b, p_b)^\top \mathrm{RoPE}(k_i, p_i) \right),$$

$$h_b = \sum_i \alpha_{b,i} v_i. \tag{10}$$

Here, $\alpha_{b,i}$ is the attention from beacon token $b$ to raw token $i$, and $p_b, p_i$ denote their RoPE positions. Since beacon states are formed through RoPE-aware QK interactions, they can absorb coarse positional cues from raw tokens even though explicit K-RoPE is removed during beacon generation.

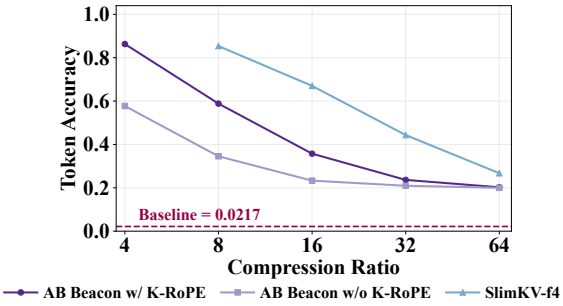


Figure 6: Reconstruction accuracy from compressed beacon states on length-16,384 random sequences over 46 symbols, averaged over three seeds. SlimKV starts from 8× total compression due to 4× feature compression. Dashed line denotes random-guessing probability.

### 4.2 Positional Asymmetry

SlimKV leverages the positional asymmetry between raw and beacon tokens. Raw tokens are lexical units aligned with token-level RoPE, whereas beacon tokens aggregate span-level context and cannot be faithfully tied to a single RoPE index. K-side RoPE ablations in Figure 2 show that removing K-RoPE degrades both raw-token and beacon Keys, but the drop is much smaller for beacon Keys. Attention maps corroborate this pattern (Figure 5): beacon K-RoPE removal largely preserves the diagonal structure, while raw-token K-RoPE removal distorts it. Table 1 quantifies this observation, with metric definitions provided in Appendix A.2. Figure 6 further shows that RoPE-free beacon states reconstruct random character sequences far above random guessing, indicating non-trivial positional cues without explicit K-RoPE.

### 4.3 Reconstruction-free Latent-Space Attention

Based on these findings, SlimKV imposes a RoPE-free constraint on beacon Keys during training, encouraging beacon states to acquire implicit positional cues during compression. With explicit beacon K-RoPE removed, SlimKV performs latent-space attention (LSA) during decoding to avoid full-Key reconstruction. For one decoding step, let $C_K, C_V \in \mathbb{R}^{N_b \times D/k}$ denote cached latent beacon K/V states, with up-projections $U_K, U_V$. If beacon Keys retain K-RoPE, attention requires

$$Q' \,\mathrm{RoPE}(K_b)^\top = Q' \,\mathrm{RoPE}(C_K U_K^\top)^\top, \quad (11)$$

where $Q'$ denotes RoPE-applied Queries. RoPE is position-dependent and cannot be absorbed into a single position-independent projection. Removing beacon K-RoPE eliminates this obstacle and allows attention to be associated in the latent space:

$$Q' K_b^\top = Q'(C_K U_K^\top)^\top = (Q' U_K) C_K^\top, \quad (12)$$

where $Q' U_K$ is computed after GQA grouping. The Value path is handled analogously:

$$A V_b = A(C_V U_V^\top) = (A C_V) U_V^\top. \quad (13)$$

Thus, LSA computes attention scores and aggregates Values directly in the latent space.

### 4.4 Cost Analysis

We analyze the dominant FLOPs per decoding step, deferring the comprehensive derivation to Appendix A.3. Let $H_q$ and $H_{\mathrm{kv}}$ denote the number of query and KV heads, $g = H_q/H_{\mathrm{kv}}$ the GQA group size, $d$ the per-head dimension, $D = H_{\mathrm{kv}} d$ the total KV dimension, $k$ the feature compression ratio, and $N_b$ the number of cached beacon tokens.

The reconstruction-based baseline with beacon K-RoPE has arithmetic cost:

$$F_{\mathrm{recon}} \approx N_b \frac{2D^2}{k} + (1 + 2g) N_b D, \quad (14)$$

where the first and second terms represent full KV reconstruction and standard attention computation, respectively. In contrast, SlimKV incurs a cost of:

$$F_{\mathrm{SlimKV}} \approx \frac{2gD^2}{k} + \frac{2H_q}{k} N_b D, \quad (15)$$

where the first and second terms correspond to per-step query/output projection overhead and latent-space attention computation.

Although both methods require $\mathcal{O}(N_b D)$ attention computation, the reconstruction baseline additionally materializes full-dimensional KV states at $\mathcal{O}(N_b D^2/k)$ cost. This sequence-dependent term can dominate long-context decoding, while SlimKV replaces it with an $\mathcal{O}(gD^2/k)$ per-step projection cost independent of $N_b$. This analysis covers arithmetic cost only, with empirical latency reported in Section 5.4.

During decoding, newly generated tokens are cached as raw tokens. This leads to a hybrid attention scheme where beacon queries use latent-space attention and raw-token queries use standard attention, as detailed in Appendix A.3.1. Since beacon states dominate long-context caches, optimizing beacon-query latency remains important.

Table 2: LongBench scores under high KV-cache compression. SlimKV `t`$n$`f`$m$: $n\times$ token-wise, $m\times$ feature-wise. The **bold** and underlined numbers denote the first and second rankings, respectively.

| Comp. | Method | LLAMA-3.1-8B | | | | | | QWEN2.5-14B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S-Doc | M-Doc | Summ. | Few | Code | Avg. | S-Doc | M-Doc | Summ. | Few | Code | Avg. |
| 1× | Full KV | 44.34 | 46.65 | 28.71 | 69.45 | 58.37 | 49.50 | 42.72 | 43.73 | 28.05 | 72.04 | 61.05 | 51.52 |
| 16× | KVZip | 33.86 | 29.11 | 23.97 | 22.97 | 34.89 | 28.96 | 7.83 | 5.48 | 3.27 | 35.12 | 14.20 | 13.18 |
| | PALU | 13.18 | 9.80 | 12.75 | 37.86 | 21.04 | 18.92 | 17.77 | 14.18 | 21.40 | 59.40 | 25.75 | 27.70 |
| | SnapKV | 38.55 | **43.30** | 23.61 | **65.92** | 56.62 | 45.60 | 36.74 | **51.20** | 20.22 | 68.50 | 52.37 | 45.81 |
| | Activation Beacon | 34.72 | 36.28 | 25.34 | 65.01 | 65.14 | 45.30 | 35.76 | 38.31 | 25.24 | 63.66 | **57.77** | 44.15 |
| | SlimKV-f2 (`t8f2`) | 37.64 | 37.40 | 26.25 | 65.11 | 64.55 | 46.19 | **39.44** | 42.16 | **25.91** | 67.20 | 57.72 | **46.48** |
| | SlimKV-f4 (`t4f4`) | **38.65** | 34.33 | **26.57** | 65.14 | **67.11** | **46.36** | 38.15 | 43.76 | 25.50 | **69.16** | 54.61 | 46.23 |
| | SlimKV-f8 (`t2f8`) | 33.46 | 36.24 | 26.02 | 65.68 | 63.82 | 45.05 | 38.25 | 38.83 | 25.61 | 68.06 | 50.89 | 44.33 |
| 32× | KVZip | 9.50 | 3.35 | 12.69 | 0.96 | 20.48 | 9.40 | 1.09 | 0.78 | 1.32 | 1.53 | 20.91 | 5.12 |
| | PALU | 1.58 | 0.56 | 5.47 | 1.32 | 15.02 | 4.79 | 4.20 | 2.82 | 10.90 | 14.66 | 13.67 | 9.25 |
| | SnapKV | **35.80** | **42.82** | 21.61 | 62.70 | 53.29 | 43.24 | 32.45 | **49.61** | 18.09 | 62.71 | 48.54 | 42.28 |
| | Activation Beacon | 29.94 | 35.24 | 24.53 | 64.26 | 64.35 | 43.67 | 32.80 | 35.64 | 25.16 | 63.20 | 57.01 | 42.76 |
| | SlimKV-f2 (`t16f2`) | 33.49 | 35.34 | 24.86 | 61.19 | 59.68 | 42.91 | 37.15 | 38.59 | 25.49 | 64.97 | **57.09** | 44.66 |
| | SlimKV-f4 (`t8f4`) | 34.16 | 35.75 | **25.43** | 63.35 | **65.07** | **44.75** | **37.55** | 40.29 | **25.50** | **68.76** | 53.36 | **45.09** |
| | SlimKV-f8 (`t4f8`) | 32.53 | 35.41 | 25.14 | **64.36** | 62.22 | 43.93 | 35.83 | 37.99 | 25.27 | 68.27 | 56.10 | 44.69 |

Table 3: LongBench gap reduction over Activation Beacon. $\Delta$ denotes the drop from Full KV.

| Backbone | Comp. | $\Delta_{\text{AB}}$ | $\Delta_{\text{SlimKV-f4}}$ | $\Delta$ Red. | Gap Red. |
|---|---|---|---|---|---|
| LLAMA-3.1-8B | 16× | 4.20 | 3.14 | 1.06 | 25.2% |
| | 32× | 5.83 | 4.75 | 1.08 | 18.5% |
| QWEN2.5-14B | 16× | 7.37 | 5.29 | 2.08 | 28.2% |
| | 32× | 8.76 | 6.43 | 2.33 | 26.6% |

Table 4: LongBench averages at low compression. The averages of the uncompressed models are in Table 2.

| Backbone | Comp. | KVZip | PALU | AB | SlimKV-f2 |
|---|---|---|---|---|---|
| LLAMA-3.1-8B | 4× | 36.39 | 46.66 | **48.21** | 48.10 |
| | 8× | 37.25 | 46.42 | 46.93 | **47.77** |
| QWEN2.5-14B | 4× | 47.72 | 45.95 | 48.86 | **49.92** |
| | 8× | 46.74 | 45.15 | 45.50 | **49.94** |

# 5 Experiments

## 5.1 Experimental Setup

**Models and training.** We evaluate SlimKV on Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-14B-Instruct (Team, 2024; Yang et al., 2024). Backbones are frozen, and training is restricted to the beacon-branch QKV projections. We follow a two-stage recipe: pre-training on 0.5B RedPajama (Weber et al., 2024) tokens, followed by fine-tuning on LongAlpaca (Chen et al., 2024), BookSum (Kryscinski et al., 2022), and GPT-generated synthetic data (Zhang et al., 2025). All samples are shorter than 20K tokens. We use batch size 8, learning rates of $5 \times 10^{-5}/1 \times 10^{-5}$, linear decay, and no warmup. We apply the auto-regressive loss only to non-beacon tokens, while beacon tokens serve as compressed memory states and are excluded from supervision.

**Compression settings.** SlimKV compresses KV cache along token and feature dimensions. Token compression is controlled by the beacon interval, and feature compression by beacon KV latent ranks, with $2\times$, $4\times$, and $8\times$ settings evaluated. Raw-token Keys and all Queries retain RoPE, while beacon Keys remove RoPE. Layer-wise ranks follow the floor-constrained equal-energy criterion in Section 3.3, with low-rank beacon projections initialized from truncated SVD of raw-token KV projections. We set $\lambda = 0.5$ to balance global rank budget and minimum layer capacity.

**Baselines.** We compare SlimKV with Full KV and representative KV-cache compression methods: PALU (Chang et al., 2025) for low-rank KV compression, KVZip (Kim et al., 2026) and SnapKV (Li et al., 2024) for KV eviction, and Activation Beacon (Zhang et al., 2025) for prompt-based soft compression. For the trainable methods, SlimKV and Activation Beacon, we use the same training data to ensure a fair comparison.

## 5.2 Effectiveness Evaluation

We evaluate SlimKV on LongBench (Bai et al., 2024) and Needle-in-a-Haystack (NIAH) (Kamradt, 2023). LongBench measures long-context performance across QA, summarization, few-shot learning, and code, while NIAH probes position-sensitive retrieval. We mainly study $16\times$ and $32\times$ compression, and additionally report $4\times$ and $8\times$ LongBench results to assess robustness across compression ratios.

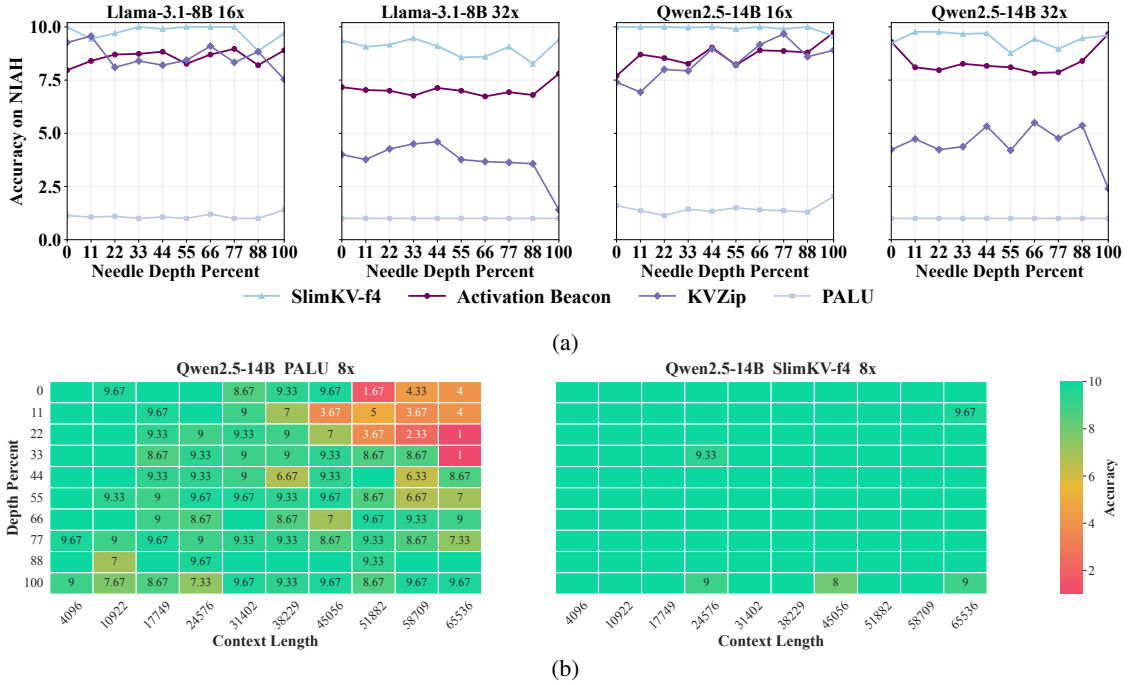


Figure 7: Needle-in-a-Haystack evaluation. (a) SlimKV outperforms baselines across depths under 16× and 32× compression. (b) SlimKV maintains retrieval ability for information beyond Qwen2.5-14B's context window (32K). Since higher ratios limit PALU's retrieval accuracy, we select 8× compression to better reveal the behavior.

**LongBench.** Table 2 reports the LongBench results. Across both backbones and high compression ratios, SlimKV achieves the best average score. Table 3 further shows that SlimKV-f4 reduces the gap to the uncompressed model by 18.5%–28.2% over Activation Beacon, demonstrating its advantage over token-only soft compression. Comparing 2×, 4×, and 8× feature compression under the same overall compression ratio reveals a trade-off: too little feature compression forces aggressive token compression, while too much feature compression restricts the latent KV capacity. The 4× setting strikes the best balance and achieves the highest average score in most high-compression settings.

At lower 4× and 8× compression ratios, SlimKV uses 2× feature compression to retain non-trivial token compression. As shown in Table 4, SlimKV preserves over 96% of the uncompressed average score, consistently outperforms KVZip and PALU, remains competitive with Activation Beacon, and achieves the best score in most low-ratio settings. This highlights SlimKV's robustness across different compression ratios.

We include SnapKV as a reference, while noting that it operates in the question-aware setting. SnapKV uses query-dependent attention patterns near the end of the prompt (Li et al., 2024), giving it a more targeted compression signal than question-agnostic methods like SlimKV. Nevertheless, SlimKV outperforms SnapKV while supporting broader scenarios such as multi-turn dialogue, streaming generation, and cache reuse. This comparison is intended to contextualize SlimKV's performance and practicality within the broader KV-cache compression landscape.

**Needle-in-a-Haystack (NIAH).** We further evaluate position-sensitive retrieval with NIAH, which is particularly relevant because removing K-RoPE from beacon may raise concerns about positional robustness (Liu et al., 2024b). NIAH results are averaged over three random seeds, using GPT-5.4-mini (OpenAI, 2026) as the judge. Figure 7a shows SlimKV-f4 outperforms baselines across backbones and compression ratios. Its stable performance across needle depths indicates robust retrieval under varying positions.

As shown in Figure 7b, when the target information lies beyond Qwen2.5-14B's context window, PALU shows reduced accuracy, whereas SlimKV maintains stable performance across context lengths and needle depths. This is partly because SlimKV compresses long context into fewer beacon tokens, keeping the cache within the usable context range. Meanwhile, the stable retrieval results show that K-RoPE-free beacon memories remain effective for beyond-window information.

Table 5: LongBench category scores on Qwen3-30B-A3B-Instruct-2507.

| Method | Comp. | S-Doc | M-Doc | Summ. | Few-shot | Code | Avg. |
|---|---|---|---|---|---|---|---|
| Full KV | 1× | 42.87 | 51.02 | 25.60 | 71.26 | 60.75 | 50.30 |
| SlimKV-f4 | 16× | 36.87 | 40.43 | 26.00 | 68.24 | 58.87 | 46.08 |
| SlimKV-f4 | 32× | 34.14 | 38.16 | 24.77 | 67.85 | 63.09 | 45.60 |

### 5.3 Scalability and Parameter Analysis

To assess scalability beyond the dense backbones used in the main evaluation, we evaluate Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025). We compare the uncompressed model with SlimKV-f4 at 16× and 32× compression, which evaluates SlimKV at a larger scale and on an MoE LLM.

Table 5 shows that SlimKV-f4 preserves competitive LongBench performance with only 434M trainable parameters, which account for 1.45% of the original model size. Because trainable-parameter storage scales with the hidden dimension, SlimKV-f4 uses fewer trainable parameters on this 30B MoE backbone than on Llama-3.1-8B and Qwen2.5-14B, which require 621M and 1.41B trainable parameters, respectively. These results demonstrate parameter-efficient scalability and preliminary transfer to larger MoE LLMs.

### 5.4 Efficiency Evaluation

We evaluate efficiency on Llama-3.1-8B at 16× and 32× compression, corresponding to `t4f4` and `t8f4`, by comparing SlimKV with a reconstruction-based variant that retains beacon K-RoPE and materializes full-dimensional K/V states before attention, with the uncompressed model as reference. We mainly report single-step attention latency and end-to-end per-token decoding latency.

Experiments use 32K–128K contexts and generate 128 tokens per run for average per-token latency. All methods use batch size 1, BF16 precision, and PyTorch eager execution on a single A100 80GB GPU. We report the median over three runs after one warm-up. Synthetic full KV-cache tensors are used for the uncompressed model to avoid prefill memory limits. Speedup are reported in Table 6.

Due to OOM at longer contexts, the uncompressed model is evaluated for Time-to-First-Token (TTFT) and peak memory at its longest feasible length of 11.5K tokens. At this length, SlimKV `t4f4`/`t8f4` achieve 1.03×/1.37× TTFT speedups over the uncompressed model and reduce peak memory by 2.71×/2.82×, respectively. Longer-context results are provided in Appendix A.5.

Table 6: Decoding speedup over the uncompressed model on Llama-3.1-8B-Instruct. Attn./E2E denote attention/per-token latency.

| Comp. | Metric | Method | 32K | 64K | 96K | 128K |
|---|---|---|---|---|---|---|
| 16× | Attn. | Recon | 1.76× | 2.81× | 3.63× | 5.34× |
| | | SlimKV | 1.94× | 3.81× | 5.08× | 7.34× |
| | E2E | Recon | 1.13× | 1.64× | 2.13× | 2.86× |
| | | SlimKV | 1.22× | 2.04× | 2.63× | 3.38× |
| 32× | Attn. | Recon | 1.52× | 2.35× | 3.84× | 4.92× |
| | | SlimKV | 1.62× | 2.54× | 4.36× | 6.09× |
| | E2E | Recon | 1.15× | 1.67× | 2.60× | 3.06× |
| | | SlimKV | 1.23× | 1.75× | 2.82× | 3.49× |

Table 7: Ablation study on LongBench at 32× compression with Llama-3.1-8B-Instruct.

| T-wise | F-wise | Train-R | Infer-R | Ada-r | Score |
|---|---|---|---|---|---|
| ✓ | ✗ | ✓ | ✓ | ✗ | 43.67 |
| ✓ | ✗ | ✓ | ✗ | ✗ | 41.28 |
| ✓ | ✗ | ✗ | ✗ | ✗ | 41.95 |
| ✓ | ✓ | ✗ | ✗ | ✗ | 44.38 |
| ✓ | ✓ | ✗ | ✗ | ✓ | 44.75 |

### 5.5 Ablation Studies

We ablate SlimKV on Llama-3.1-8B-Instruct under 32× total compression. Token-only variants use 32× token-wise compression, while joint variants use 8× token-wise and 4× feature-wise (`t8f4`). T-wise/F-wise denote token and feature compression, Train-R/Infer-R denote beacon K-RoPE during training/inference, and Ada-r denotes layer-adaptive ranks.

Removing beacon K-RoPE does degrade accuracy, showing that RoPE removal alone is insufficient. SlimKV instead trains beacon projections directly in a low-rank form under the no-K-RoPE constraint. Layer-adaptive rank allocation further improves performance, and the full design ultimately surpasses Activation Beacon.

## 6 Conclusion

We introduced SlimKV, a question-agnostic joint token-feature KV-cache compression framework for long-context LLMs. SlimKV learns layer-adaptive low-rank beacon KV representations and removes key-side RoPE only from beacon Keys, enabling reconstruction-free latent-space attention. Experiments on LongBench, Needle-in-a-Haystack, and decoding efficiency demonstrate a stronger accuracy-efficiency trade-off than prior KV-cache compression methods.

## Limitations

First, SlimKV is mainly evaluated on RoPE-based language models, which represent a widely used family of current LLM backbones. Since our reconstruction-free design addresses the key-side RoPE reconstruction bottleneck in low-rank attention, its behavior under other positional encoding schemes (Raffel et al., 2020; Press et al., 2022; Peng et al., 2024) remains to be further examined.

Second, our experiments focus on text-only long-context language models. Extending SlimKV to broader architectures, such as multimodal LLMs with visual or cross-modal KV caches, is an interesting direction but requires additional validation.

## References


Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137. Association for Computational Linguistics.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. 2025. Palu: KV-cache compression with low-rank projection. In *The Thirteenth International Conference on Learning Representations*.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. LongloRA: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations*.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*.

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. *Preprint*, arXiv:2407.21783.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. 2025. Towards economical inference: Enabling DeepSeek's multi-head latent attention in any transformer-based LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33313–33328. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677. Association for Computational Linguistics.

Greg Kamradt. 2023. Needle in a haystack: Pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. 2026. KVzip: Query-agnostic KV cache compression with context reconstruction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Wojciech Kryscinski, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. 2022. BOOKSUM: A collection of datasets for long-form narrative summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6536–6558. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zongqian Li, Yixuan Su, and Nigel Collier. 2025. 500xCompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25081–25091. Association for Computational Linguistics.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024b. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*.

OpenAI. 2026. Introducing gpt-5.4 mini and nano. `https://openai.com/index/introducing-gpt-5-4-mini-and-nano/`. Accessed: 2026-05-26.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, volume 2024, pages 31932–31951.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15332–15344. Association for Computational Linguistics.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2023. Focused transformer: Contrastive training for context scaling. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Maurice Weber, Daniel Y Fu, Quentin Gregory Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Re, Irina Rish, and Ce Zhang. 2024. Redpajama: an open dataset for training large language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2025. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. Cam: Cache merging for memory-efficient LLMs inference. In *Forty-first International Conference on Machine Learning*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

# A Appendix

## A.1 Quantitative Analysis of Layer-wise Rank Consistency

We provide additional quantitative evidence for the layer-wise rank consistency between Beacon and raw K/V projections. For each target compression ratio, ranks are selected using the same explained-variance criterion under an identical average rank budget. In Table 8, Beacon and Raw report the average selected ranks, Spearman measures whether the two rank profiles induce the same layer ordering, and MAE / Max reports the mean and maximum absolute per-layer rank differences.

Table 8: Layer-wise rank consistency between Beacon and raw K/V projections.

| Model | Proj. | CR | Beacon | Raw | Spearman | MAE / Max |
|---|---|---|---|---|---|---|
| LLAMA | K | 2 | 64.00 | 64.00 | 1.000 | 0.00 / 0 |
| | K | 4 | 32.00 | 32.00 | 0.996 | 0.06 / 1 |
| | K | 8 | 16.00 | 16.00 | 0.996 | 0.06 / 1 |
| | V | 2 | 64.00 | 64.00 | 0.990 | 0.50 / 1 |
| | V | 4 | 32.00 | 32.00 | 0.954 | 0.50 / 2 |
| | V | 8 | 16.00 | 16.00 | 0.928 | 0.38 / 2 |
| QWEN | K | 2 | 64.00 | 64.00 | 0.997 | 0.08 / 1 |
| | K | 4 | 32.00 | 32.00 | 0.999 | 0.04 / 1 |
| | K | 8 | 16.00 | 16.00 | 0.997 | 0.04 / 1 |
| | V | 2 | 64.00 | 64.00 | 0.997 | 0.08 / 1 |
| | V | 4 | 32.00 | 32.00 | 1.000 | 0.00 / 0 |
| | V | 8 | 16.00 | 16.00 | 0.990 | 0.04 / 1 |

The results show strong agreement between Beacon and raw projections across models, projections, and compression ratios. In particular, the rank profiles achieve consistently high Spearman correlations, while the absolute discrepancies remain within one or two ranks in almost all cases. This supports using raw K/V projection spectra as a reliable proxy for layer-adaptive rank allocation before Beacon training.

## A.2 Interpretation of Attention Similarity Metrics

Figure 5 visualizes the K-RoPE ablations, and Table 1 quantifies their similarity to the K-Kept setting. Let $A$ and $B$ be two averaged attention maps. All metrics are computed on the valid causal region

$$\Omega = \{(i, j) : j \leq i\}. \tag{16}$$

**Attention-map Pearson.** We flatten $A$ and $B$ within $\Omega$ and compute

$$\rho_{\text{attn}} = \text{corr}\left(\{A_{ij}\}_{(i,j)\in\Omega}, \{B_{ij}\}_{(i,j)\in\Omega}\right). \tag{17}$$

Higher values indicate more similar global attention patterns.

**Coarse-block Pearson.** We average valid entries within coarse blocks to obtain $\widetilde{A}$ and $\widetilde{B}$, and compute

$$\rho_{\text{block}} = \text{corr}\left(\text{vec}(\widetilde{A}), \text{vec}(\widetilde{B})\right). \tag{18}$$

Higher values indicate more similar macroscopic structures.

**Distribution Divergence.** For the two profile-based metrics below, we use Jensen-Shannon divergence:

$$\text{JS}(p, q) = \frac{1}{2}\text{KL}(p\|m) + \frac{1}{2}\text{KL}(q\|m), \tag{19}$$

$$m = \frac{1}{2}(p + q), \tag{20}$$

where $\text{KL}(p\|m) = \sum_x p(x) \log \frac{p(x)}{m(x)}$. Lower JS values indicate more similar distributions.

**Lag-profile JS Divergence.** We aggregate attention mass by relative distance $\ell = i - j$ and normalize it into lag profiles $p_A(\ell)$ and $p_B(\ell)$:

$$D_{\text{lag}} = \text{JS}(p_A, p_B). \tag{21}$$

Lower values indicate more similar near-range and long-range attention allocation.

**Column-profile JS Divergence.** We aggregate attention mass by key position $j$ and normalize it into column profiles $q_A(j)$ and $q_B(j)$:

$$D_{\text{col}} = \text{JS}(q_A, q_B). \tag{22}$$

Lower values indicate more similar attended-key distributions.

## A.3 Derivation of Decoding Cost

We derive the dominant FLOPs of one decoding step. Let $H_q$ and $H_{kv}$ denote the numbers of query and KV heads, respectively, and let $g = H_q/H_{kv}$ be the GQA group size. Let $d$ be the per-head dimension, $D = H_{kv}d$ the concatenated KV dimension, $k$ the feature-wise compression ratio, and $N_b$ the number of cached beacon tokens. We consider a multi-head-coupled low-rank setting, where the latent KV dimension is $D/k$ over the concatenated KV space.

**Reconstruction-based baseline.** The reconstruction baseline stores the compressed beacon KV states $C_K, C_V \in \mathbb{R}^{N_b \times D/k}$ in the cache, and then expands them with the up-projections $U_K, U_V \in \mathbb{R}^{D/k \times D}$ before attention:

$$K = C_K U_K, \qquad V = C_V U_V. \tag{23}$$

Reconstructing both Keys and Values costs

$$2 \cdot N_b \frac{D}{k} D = \frac{2N_b D^2}{k}. \tag{24}$$

After reconstruction, standard GQA attention is performed in the full feature space. The score computation and Value aggregation each cost $H_q N_b d = g N_b D$. In addition, applying K-side RoPE to reconstructed beacon Keys costs $N_b D$. Therefore,

$$F_{\text{recon}} \approx N_b \frac{2D^2}{k} + (1 + 2g) N_b D. \tag{25}$$

**SlimKV.** SlimKV removes explicit K-side RoPE from beacon Keys and avoids full KV reconstruction. For each query head, the query is first projected into the latent space:

$$\tilde{q}_h = q_h U_{K,h}^\top \in \mathbb{R}^{D/k}. \tag{26}$$

This costs $dD/k$ per query head, or $H_q dD/k = gD^2/k$ over all query heads. The value-side expansion from the latent space back to the query-head output space has the same cost, giving a local projection cost of $2gD^2/k$.

Attention is then computed directly in the latent space. For each query head, both the score computation $\tilde{q}_h C_K^\top$ and the weighted aggregation over $C_V$ cost $N_b D/k$. Thus the latent attention cost is

$$2 H_q N_b \frac{D}{k} = \frac{2H_q}{k} N_b D. \tag{27}$$

Combining the local projection and latent attention terms yields

$$F_{\text{SlimKV}} \approx \frac{2gD^2}{k} + \frac{2H_q}{k} N_b D. \tag{28}$$

The key difference is that the reconstruction-based baseline incurs an $N_b$-scaled full-KV reconstruction cost, whereas SlimKV applies low-rank transformations only to the current-step query/output and keeps the dominant attention computation in the compressed latent space.

**Typical-value analysis.** We further instantiate Equations (25) and (28) with a representative GQA configuration, $H_q = 32$, $H_{kv} = 8$, $g = 4$, $k = 4$, and $D = 1024$. For the reconstruction-based baseline, this gives

$$F_{\text{recon}} \approx \frac{1}{2} N_b D^2 + 9 N_b D \approx 521 N_b D, \tag{29}$$

where one factor of $D$ in the reconstruction term is instantiated as 1024. For SlimKV, we have

$$F_{\text{SlimKV}} \approx 2D^2 + 16 N_b D. \tag{30}$$

The first term only comes from current-step query/output transformations and does not scale with $N_b$. It is dominated by the latent attention term when $2D^2 \ll 16 N_b D$, i.e., $N_b \gg D/8$. For $D = 1024$, this requires $N_b \gg 128$, which is typically satisfied in long-context decoding. Therefore, we approximate

$$F_{\text{SlimKV}} \approx 16 N_b D. \tag{31}$$

Under this typical setting, the dominant arithmetic reduction is

$$\frac{F_{\text{recon}}}{F_{\text{SlimKV}}} \approx \frac{521 N_b D}{16 N_b D} \approx 32.6 \times . \tag{32}$$

We emphasize that this comparison only reflects theoretical arithmetic cost: lower-order operations, kernel launch overheads, memory layout effects, and cache/memory-bandwidth limitations are omitted. Therefore, the arithmetic reduction should be interpreted as explaining the computational scaling trend rather than directly predicting measured wall-clock decoding speedup.

#### A.3.1 Hybrid Attention with Raw-Token Tail

In practical decoding, the cache may contain both compressed beacon tokens and a short tail of uncompressed raw tokens, e.g., newly generated tokens after the last compression step. We therefore use a hybrid attention computation: beacon tokens are processed in the latent KV space, while raw tokens are processed in the original feature space.

For a query head $h$, let $Q_h \in \mathbb{R}^{n \times d}$ be the current queries, $K_{r,h}, V_{r,h} \in \mathbb{R}^{N_r \times d}$ be the raw-token KV cache, and $C_K, C_V \in \mathbb{R}^{N_b \times D/k}$ be the latent beacon KV cache. Let $U_{K,h}, U_{V,h} \in \mathbb{R}^{D/k \times d}$ denote the corresponding low-rank up-projection matrices. We compute beacon and raw logits as

$$S_{b,h} = \frac{(Q_h U_{K,h}^\top) C_K^\top}{\sqrt{d}}, \qquad S_{r,h} = \frac{Q_h K_{r,h}^\top}{\sqrt{d}}. \tag{33}$$

The two logits are concatenated before softmax:

$$[A_{b,h}, A_{r,h}] = \mathrm{Softmax}\left([S_{b,h}, S_{r,h}]\right), \tag{34}$$

so beacon and raw tokens are normalized under the same attention distribution. The output is then

$$O_h = (A_{b,h}C_V)U_{V,h} + A_{r,h}V_{r,h}. \tag{35}$$

This is mathematically equivalent to reconstructing beacon KV states as $K_{b,h} = C_K U_{K,h}$ and $V_{b,h} = C_V U_{V,h}$, concatenating them with raw-token KV states, and applying standard attention:

$$O_h = \mathrm{Softmax}\left(\frac{Q_h[K_{b,h}; K_{r,h}]^\top}{\sqrt{d}}\right)[V_{b,h}; V_{r,h}]. \tag{36}$$

The equivalence follows directly from associativity, i.e., $Q_h(C_K U_{K,h})^\top = (Q_h U_{K,h}^\top)C_K^\top$ and $A_{b,h}(C_V U_{V,h}) = (A_{b,h}C_V)U_{V,h}$. Thus, hybrid attention preserves the exact standard attention result while avoiding explicit full-dimensional reconstruction for beacon tokens.

### A.4 Implementation Details

**Training cost and hardware.** SlimKV freezes the backbone and updates only the beacon-branch QKV projections. All training runs are conducted with BF16 precision on 4 NVIDIA A100 80GB GPUs. For Llama-3.1-8B-Instruct, pre-training takes approximately 8 hours and fine-tuning takes approximately 6 hours, with about 50GB peak GPU memory usage. For Qwen2.5-14B-Instruct, pre-training takes approximately 14 hours and fine-tuning takes approximately 11 hours, with about 78GB peak GPU memory usage.

**PALU settings.** PALU jointly compresses KV cache by reducing feature dimension and numerical precision, so its total compression depends on both factors. We follow the original PALU settings whenever the target compression ratio is available. Otherwise, we choose the closest original setting and decrease feature compression only when necessary to meet the target ratio. This gives PALU a conservative configuration, since we observe that PALU is more sensitive to feature-dimensional compression than bit-width reduction. For 4×, 8×, 16×, and 32× total compression, we use 1.5×/6-bit, 1.5×/3-bit, 2×/3-bit, and 4×/2-bit feature/precision settings, respectively.

### A.5 Detailed Efficiency Results

**Decoding latency.** Table 9 reports absolute decoding latency over 128 generated tokens. Recon denotes the reconstruction-based variant that retains beacon K-RoPE and materializes full-dimensional K/V states before attention. SlimKV consistently reduces both attention and end-to-end latency, confirming the benefit of reconstruction-free latent-space attention.

Table 9: Detailed decoding latency on Llama-3.1-8B-Instruct over 128 generated tokens. Attn. and E2E denote cumulative attention and end-to-end decoding latency. All values are in seconds.

| Comp. | Method | 32K | | 64K | | 96K | | 128K | |
|---|---|---|---|---|---|---|---|---|---|
| | | Attn. | E2E | Attn. | E2E | Attn. | E2E | Attn. | E2E |
| 1× | Full KV | 11.84 | 13.28 | 23.97 | 25.40 | 34.93 | 36.32 | 46.64 | 48.08 |
| 16× | Recon | 6.74 | 11.75 | 8.52 | 15.46 | 9.63 | 17.02 | 8.73 | 16.79 |
| | SlimKV | 6.11 | 10.93 | 6.29 | 12.47 | 6.87 | 13.79 | 6.35 | 14.24 |
| 32× | Recon | 7.80 | 11.56 | 10.18 | 15.23 | 9.11 | 13.95 | 9.48 | 15.69 |
| | SlimKV | 7.31 | 10.81 | 9.43 | 14.51 | 8.01 | 12.90 | 7.65 | 13.78 |

**TTFT and peak memory.** Table 10 reports TTFT and PyTorch peak allocated memory. The uncompressed model runs out of memory at longer contexts, so the main text reports its longest feasible length of 11.5K tokens. Here we provide detailed measurements for SlimKV and the reconstruction-based variant at longer context lengths.

Table 10: TTFT and peak allocated memory on Llama-3.1-8B-Instruct. TTFT is reported in seconds and memory in GB.

| Comp. | Method | 32K | 64K | 96K | 128K |
|---|---|---|---|---|---|
| **TTFT (s)** | | | | | |
| 16× | Recon | 16.59 | 46.38 | 89.15 | 153.74 |
| | SlimKV | 16.16 | 44.80 | 87.12 | 152.57 |
| 32× | Recon | 11.28 | 28.34 | 50.13 | 82.73 |
| | SlimKV | 10.92 | 27.77 | 48.62 | 79.96 |
| **Peak Memory (GB)** | | | | | |
| 16× | Recon | 25.73 | 29.77 | 33.80 | 37.84 |
| | SlimKV | 25.72 | 29.76 | 33.80 | 37.84 |
| 32× | Recon | 23.42 | 25.26 | 27.09 | 28.92 |
| | SlimKV | 23.41 | 25.25 | 27.08 | 28.91 |

Table 10 shows that higher compression ratios lead to clearly lower peak memory usage. SlimKV and the reconstruction-based variant have nearly identical peak memory because reconstruction changes only the computation flow, not the stored KV-cache size. SlimKV also achieves con-

sistent but modest TTFT improvements over the reconstruction-based variant, mainly by avoiding key-side RoPE computation and full-Key materialization for beacon states.

### A.6 Task-wise Analysis on LongBench

We further analyze SlimKV across LongBench task categories. As shown in Table 2, SlimKV-f4 brings the most consistent gains on document QA tasks, including S-Doc and M-Doc, suggesting that RoPE-free beacon memory preserves key semantic evidence for retrieval-style reasoning while reducing token-wise compression pressure under the same total budget.

On summarization tasks, SlimKV gives smaller but stable gains, consistent with their stronger dependence on global semantic aggregation than exact token positions. Few-shot and code tasks show more backbone-dependent trends: SlimKV improves clearly on Qwen2.5-14B, while gains on Llama-3.1-8B are smaller; code tasks are also less uniform, likely due to their stronger reliance on local order, formatting, and structural cues.

Overall, RoPE-free beacon memory is especially effective for semantic aggregation and retrieval-oriented tasks, while fine-grained order-sensitive tasks remain more sensitive to the compression design.

### A.7 Needle-in-a-Haystack Evaluation Details

Following the GPT-as-judge evaluation protocol in Activation Beacon (Zhang et al., 2025), we use GPT-5.4-mini (OpenAI, 2026) as the judge with the following scoring prompt:

> Score 1: The answer is completely unrelated to the reference.
> Score 3: The answer has minor relevance but does not align with the reference.
> Score 5: The answer has moderate relevance but contains inaccuracies.
> Score 7: The answer aligns with the reference but has minor omissions.
> Score 10: The answer is completely accurate and aligns perfectly with the reference.
> Only respond with a numerical score.

Figure 8 provides the complete Needle-in-a-Haystack heatmaps across context lengths and needle depths, averaged over seeds 42, 43, and 44. SlimKV-f4 denotes 4× feature-wise compression. These results complement the main-text line plots and show that SlimKV-f4 maintains more stable retrieval performance than Activation Beacon.

### A.8 Licenses and Intended Use of Artifacts

All backbones, training corpora, and evaluation benchmarks used in this work are publicly available, and we use them in accordance with their respective licenses, terms of use, and intended research purposes. Specifically, Llama-3.1-8B-Instruct is used under the Llama 3.1 Community License; Qwen2.5-14B-Instruct and Qwen3-30B-A3B-Instruct-0527 models, RedPajama under Apache 2.0; LongAlpaca under CC-BY-NC 4.0; BookSum under BSD-3-Clause; LongBench and Needle-in-a-Haystack under MIT. GPT-5.4-mini is accessed via the OpenAI API as an LLM-as-judge for NIAH evaluation, under the standard OpenAI Terms of Use.

Figure 8: Complete Needle-in-a-Haystack heatmaps for Activation Beacon and SlimKV-f4, where f4 denotes 4× feature-wise compression. Results are shown on Llama-3.1-8B-Instruct and Qwen2.5-14B-Instruct under 16× and 32× total compression. Each cell reports retrieval accuracy at a given context length and needle depth.