AURA-KV: Adaptive Subspace Rank Approximation for Efficient Long-Context LLM Inference

Anonymous ACL submission

Abstract

KV caching is essential for efficient autoregressive decoding, but its memory footprint grows linearly with context length and quickly becomes a bottleneck for long-context large language models (LLMs). We study whether feature-dimension compression can serve as an orthogonal second stage for KV-cache reduction, and introduce AURA-KV, an adaptive-rank subspace approximation method for KV caches. AURA-KV is training-free: it decomposes post-RoPE key and value caches into compact bases and coefficients, supports incremental decoding, and allocates rank adaptively across layers under a global memory budget. Because it compresses the feature dimension rather than selecting tokens or lowering precision, it can be combined with token eviction and quantization methods. Across LongBench, RULER, and InfiniteBench, AURA-KV preserves long-context performance on Llama-3-8B, Mistral-7B, and LongChat-7B under moderate compression and degrades gracefully under more aggressive budgets. In latency measurements, AURA-KV improves per-layer decoding speed by up to $2.75\times$, and by up to $3.23\times$ when combined with token-level cache reduction, with a small one-time prefill overhead.

1 Introduction

Long-context language models are increasingly used for tasks such as multi-document question answering, repository-level code completion, retrieval-augmented generation, and long-form summarization. These applications require the model to keep thousands of previous tokens available during generation. KV caching makes such generation practical by storing past keys and values instead of recomputing them, but it also creates a state whose size grows linearly with context length, number of layers, and number of attention heads (Vaswani et al., 2017; Fu, 2024). As a result, long-context serving is often limited not by model parameters alone, but by the memory footprint and bandwidth cost of repeatedly reading the KV cache during decoding.

Existing work reduces this cost from several directions. Optimized attention kernels improve memory access and compute utilization (Dao et al., 2022; Dao, 2023); weight quantization reduces parameter memory (Frantar et al., 2022; Kim et al., 2023; Xiao et al., 2023a; Yao et al., 2022; Dettmers et al., 2022; Lin et al., 2024); sparse or approximate attention reduces the cost of long-range attention (Han et al., 2023). KV-cache compression is complementary because it directly targets the persistent state used at every decoding step. Most existing KV-cache compressors operate along either the sequence dimension or the precision dimension: token-eviction methods retain selected cached positions (Zhang et al., 2023; Xiao et al., 2023b; Li et al., 2024; Cai et al., 2024), while quantization stores cached activations at lower precision (Hooper et al., 2024; Liu et al., 2024; Kang et al., 2024). Feature-dimension compression is a natural third axis: instead of deciding which tokens to keep or how many bits to use, it asks whether each cached vector can be represented in a lower-dimensional subspace.

This direction is attractive but technically delicate. First, keys in modern decoder-only LLMs are position-dependent because rotary position embeddings (RoPE) are applied before attention; compressing the wrong representation can require repeated reconstruction or additional positional transformations during decoding. Second, the cache is not equally compressible across layers: some layers need more rank to preserve the same spectral energy, while others have much more concentrated singular-value spectra. Third, a useful compressor must support incremental decoding, where new keys and values arrive one token at a

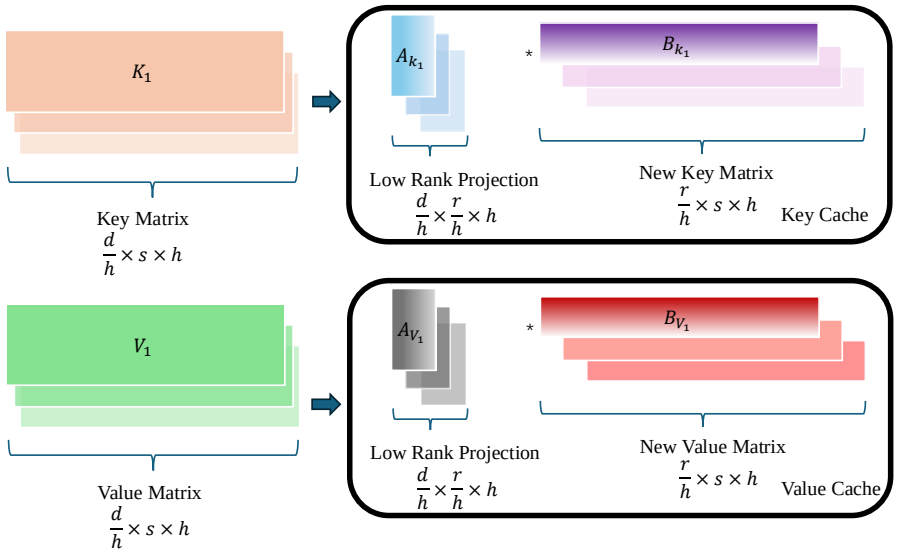


Figure 1: **Overview of AURA-KV**. For each attention head, we approximate the post-RoPE key and value caches with low-rank factors. The basis matrices map cached features into a lower-dimensional subspace, while the coefficient matrices store token-wise representations in that subspace.

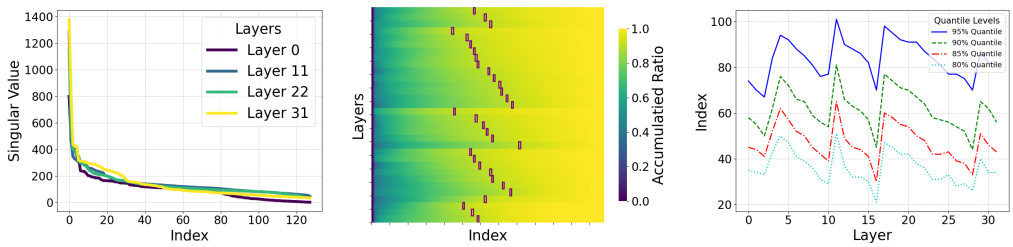
time and must be appended without recomputing a full factorization. These constraints make a naive low-rank projection insufficient for practical long-context inference.

AURA-KV addresses these constraints by treating KV-cache compression as adaptive subspace approximation after RoPE has already been applied. The name AURA-KV stands for **A**daptive s**U**bspace **R**ank **A**pproximation for KV caches. Our starting observation is that post-RoPE key and value caches have concentrated singular-value spectra: a relatively small number of components captures most of the Frobenius energy, and the required rank varies substantially across layers. This suggests that a fixed feature dimension is wasteful, but also that a single uniform rank schedule is too crude. Instead, AURA-KV selects layer-wise ranks under a global memory budget, spending rank where the cache spectra indicate it is most needed.

AURA-KV has three components. First, it factorizes each post-RoPE per-head key and value cache into an orthonormal basis and a low-dimensional coefficient matrix, allowing attention to run directly in the compressed representation. Second, it appends newly generated keys and values by projecting them into the stored bases, so the cache continues to grow with the selected rank rather than the full head dimension. Third, it performs adaptive layer-wise rank allocation by maximizing retained spectral energy subject to a global cache-memory budget. We also provide an idealized cone-effect analysis to motivate why representations can become more spectrally concentrated across layers; the theory is used as motivation for adaptive allocation, while the main evidence comes from empirical ablations and long-context evaluation. Our contributions can be summarized as follows:

- We introduce AURA-KV, a training-free post-RoPE KV-cache compressor that reduces the feature dimension of cached keys and values while supporting incremental decoding.
- We propose an adaptive layer-wise rank allocation strategy, motivated by observed spectral concentration and an idealized cone-effect analysis, to use a fixed memory budget more effectively than uniform rank allocation.
- We show on long-context benchmarks and latency measurements that AURA-KV preserves quality under moderate budgets, decodes faster than low-rank baselines, and composes naturally with token eviction and quantization.

Appendix A gives the proof of the proposition; the appendix also includes the full pseudo-code in Appendix B, implementation and memory accounting in Appendices O and I, and extended experiments in Appendices K, M, N, and Q.



(a) (b) (c)

Figure 2: **Layer-wise singular-value spectra of cached keys.** Fig. 2(a) plots the singular values of K at Layers 0, 11, 22, and 31. Fig. 2(b) shows the spectrum heatmap, where purple boxes mark the ranks that retain 90% of the Frobenius energy. Fig. 2(c) reports the ranks needed to reach 80%, 85%, 90%, and 95% Frobenius-energy thresholds across layers.

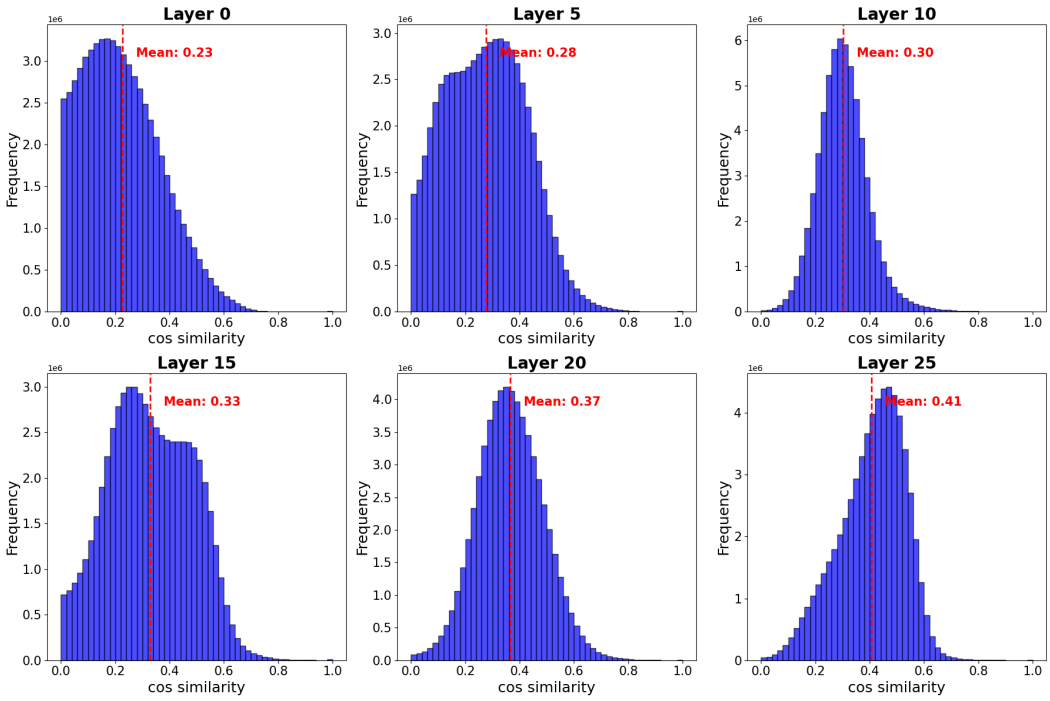


Figure 3: **Layer-wise cosine similarity of token representations.** The six panels show pairwise cosine similarities among layer inputs at Layers 0, 5, 10, 15, 20, and 25. The upward shift in deeper layers indicates increasing angular concentration, which motivates lower-rank approximations of later-layer caches.

2 Empirical Observation and Idealized Analysis

In this section, we first present our observations on the low-rank properties of the key and value matrices within transformer layers. We then introduce an idealized cone-effect analysis that motivates why later layers tend to be more compressible.

2.1 Low-Rank Structure of the KV Cache

We first inspect the singular-value spectra of cached keys and values in Llama-3-8B. As shown in Fig. 2(a), only a small subset of singular values is large, while the remaining spectrum decays rapidly. The heatmap in Fig. 2(b) shows that many layers can preserve 90% of the Frobenius energy with substantially fewer than the full feature dimension, and Fig. 2(c) reports the rank required to retain 80%, 85%, 90%, and 95% energy across layers. These observations indicate exploitable low-rank structure in the KV cache. The required rank also decreases non-uniformly across layers, suggesting that a single uniform rank wastes capacity in some layers while over-compressing others. This motivates an adaptive allocation strategy: given a fixed total memory budget, assign higher rank to layers with flatter spectra and lower rank to layers with stronger spectral concentration. The next subsection provides an idealized explanation for this broad trend.

2.2 Idealized Cone Analysis

The empirical spectra in Fig. 2 suggest that deeper layers often require fewer dimensions to preserve the same amount of energy. We connect this trend to a cone-effect view of Transformer representations.

The following proposition studies a simplified self-attention layer with Gaussian projection matrices. It should be read as an explanatory model for the observed compression trend, not as a full description of all practical Transformer components such as residual connections, LayerNorm, and MLP blocks. The compression algorithm itself does not require the Gaussian assumption; ranks are selected from empirical cache spectra, and the proposition is used only to motivate why such spectra may become concentrated. The independent-Gaussian setup follows a standard random-network analysis style used in mean-field studies of signal propagation (Poole et al., 2016), but trained LLM weights can contain strong correlations that fall outside these assumptions. Therefore, the proposition is not a rigorous proof for arbitrary trained Transformers.

Proposition 2.1. *Suppose* $X^{(\ell)} = [x_1^{(\ell)}, \dots, x_s^{(\ell)}] \in \mathbb{R}^{d\times s}$ *denotes the input token representations at layer* ℓ *where* s *is the sequence length, such that* $\|x_i^{(\ell)}\| = 1, \forall i \in \{1, 2, \dots, s\}$. $W_q, W_k, W_v \in \mathbb{R}^{d\times d}$ *are random weight matrices (*$\ell-$*th layer Q, K, V, weight matrix) where each element* $W_{a,m,n} \sim \mathcal{N}(0, d^{-2})$*, for* $a \in \{q, k, v\}, m, n \in \{1, 2, \dots, d\}$. *Then the following holds with probability at least 1-*$\mathcal{O}(1/d)$

$$|\cos\langle x_i^{(\ell+1)}, x_j^{(\ell+1)}\rangle| \geq |\cos\langle \bar{x}_i^{(\ell)}, \bar{x}_j^{(\ell)}\rangle| - |\mathcal{O}(1/d)|\,.$$

where $\bar{x}_i^{(\ell)} = \left(\sum_{\hat{i}=1}^{i} x_{\hat{i}}^{(\ell)}\right)/i, \bar{x}_j^{(\ell)} = \left(\sum_{\hat{j}=1}^{j} x_{\hat{j}}^{(\ell)}\right)/j$*, respectively.*

The proof is in Appendix A. The proposition says that, under this simplified model, the angular relation between token representations is lower-bounded by the angular relation between prefix averages, up to a small high-dimensional error term. This provides one possible explanation for why token representations become increasingly anisotropic and why later-layer KV caches may admit stronger low-rank approximation. Consistent with this view, Fig. 3 shows increasing cosine similarity across layers, while Fig. 2(b) and Fig. 2(c) show that fewer dimensions are often needed in later layers. This analysis does not explain the non-monotonic or piecewise changes in the rank curve by itself; those details are empirical properties of the measured spectra. We therefore use the cone-effect analysis only as motivation for adaptive rank allocation, and rely on the ablations in Section 4 and Appendix P to validate its empirical utility.

Moreover, to check whether the Gaussian marginal assumption is at least empirically plausible, we quantify distributional divergence with the Wasserstein distance. A small distance value signals a close alignment between the two distributions (Panaretos and Zemel, 2018). Figure 4 displays the Wasserstein distance for the weight coefficients of W_q, W_k, and W_v in every layer of Llama (Dubey et al., 2024) and Mistral (Jiang et al., 2023). The empirical distributions are compared against the Gaussian reference $\mathcal{N}(0, d^{-2})$. The observed distances are on the order of 10^{-3}, indicating close marginal agreement with the reference distribution. This check is not intended to prove the proposition's assumptions for all Transformer variants; it only shows that the simplified model is not grossly inconsistent with the inspected checkpoints. Moreover, Appendix D presents the same analysis for the LongChat (Li et al., 2023) model, and the resulting values show similar marginal behavior across multiple model architectures.

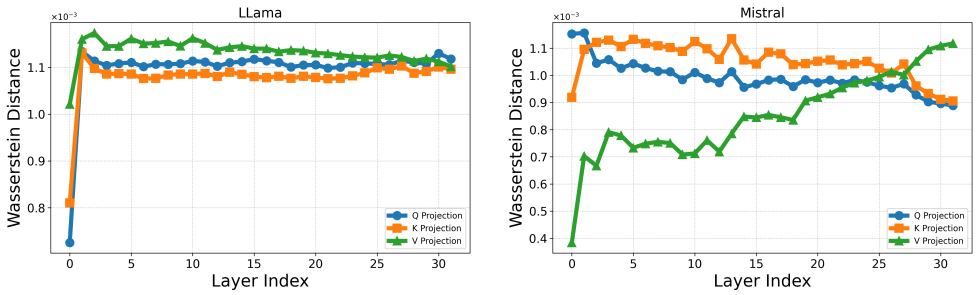
3 AURA-KV Algorithm

Our goal is to compress the KV cache to a target ratio p_{Comp} while retaining as much spectral energy as possible. For each layer ℓ and cache type $c \in \{K, V\}$, we choose rank $r_{\ell,c}$ so that the leading singular values of the cached key/value matrices preserve at least an information ratio p_{Info}:

$$\frac{\sum_{i=1}^{r_{\ell,c}} (\sigma_{\ell,c}^i)^2}{\|X_{\ell,c}\|_{\text{Fro}}^2} \geq p_{\text{Info}}, \qquad \forall \ell = 1, \dots, L, \quad c \in \{K, V\}. \tag{1}$$

where $X_{\ell,c}$ is the post-RoPE cache matrix used for calibration, $\sigma_{\ell,c}^i$ is its i-th singular value, and L is the number of Transformer layers. We search for the largest feasible p_{Info} subject to the global rank budget:

$$\max_{p_{\text{Info}}, \{r_{\ell,c}\}} \quad p_{\text{Info}} \qquad \text{s.t.} \quad \text{Eq. (1)}, \ \frac{\sum_{\ell,c} r_{\ell,c}}{2Ld_h} \leq p_{\text{Comp}}, \tag{2}$$



(a) (b)

Figure 4: **Wasserstein distance between weight distributions of W_q, W_k, W_v and $\mathcal{N}(0, d^{-2})$.** Fig. 4(a) depicts the Wasserstein distances between W_q, W_k, W_v and $\mathcal{N}(0, d^{-2})$ across all layers of Llama-3-8B-Instruct, whereas Fig. 4(b) presents the corresponding distances for Mistral-7B-Instruct.

where d_h is the per-head feature dimension. This rank-budget form is the long-context approximation of the exact cache-memory ratio because the coefficient matrices dominate the small basis matrices when $s \gg d_h$. In practice, we perform a grid search over p_{Info}, derive the minimum key/value rank for each layer, and keep the largest value that satisfies the budget. Appendix P gives the full procedure, and Appendix O details the randomized low-rank solver used after prefill.

We next describe per-head compression after the layer-wise ranks are chosen. Let $d_h = d/h$ be the feature dimension of each attention head. For head i, the standard cache stores $K_i, V_i \in \mathbb{R}^{s \times d_h}$. Given a new query $q_i \in \mathbb{R}^{1 \times d_h}$, standard attention computes

$$P_i = q_i K_i^T, \quad O_i = \text{softmax}\left(\frac{P_i}{\sqrt{d_h}}\right) V_i. \tag{3}$$

The outputs of all heads are then concatenated.

LLM inference is commonly separated into a prefill stage and a decode stage. The prefill stage is computationally intensive, while the decode stage is mainly limited by memory and bandwidth. Our algorithm optimizes the KV cache in the decode stage. Exploiting the empirical low-rank structure of the prefill cache, we approximate the transposed key and value caches as

$$K_i^T \approx A_{K_i} B_{K_i}, \quad V_i^T \approx A_{V_i} B_{V_i}, \tag{4}$$

where $A_{K_i}, A_{V_i} \in \mathbb{R}^{d_h \times r_i}$ have orthonormal columns, $B_{K_i}, B_{V_i} \in \mathbb{R}^{r_i \times s}$ store token-wise coefficients, and $r_i \ll d_h$. Appendix H summarizes the SVD notation underlying this factorization. The compression ceiling is bounded by d_h, so AURA-KV is best viewed as a complementary feature-dimension compressor; Appendix S discusses this scope. The process is shown in Fig. 1.

The compressed attention computation becomes

$$P_i = (q_i A_{K_i}) B_{K_i},$$
$$O_i = \left[\text{softmax}\left(\frac{P_i}{\sqrt{d_h}}\right) B_{V_i}^T\right] A_{V_i}^T.$$

For a newly generated token t, we append only low-dimensional coefficients to the cache:

1. **Split by head.** Decompose the new key and value into head-specific vectors:

$$k_t = \textbf{concatenate}(k_{1,t}, k_{2,t}, \ldots, k_{h,t}),$$
$$v_t = \textbf{concatenate}(v_{1,t}, v_{2,t}, \ldots, v_{h,t});$$

2. **Project into the low-rank bases.** Because A_{K_i} and A_{V_i} are orthonormal, the least-squares coefficients reduce to simple projections:

$$b_{K_i,t} = A_{K_i}^T k_{i,t}, \qquad b_{V_i,t} = A_{V_i}^T v_{i,t}.$$

3. **Append coefficients.** Update the compressed caches:

$$B_{K_i,t} = \textbf{concatenate}(B_{K_i,t-1}, b_{K_i,t}),$$
$$B_{V_i,t} = \textbf{concatenate}(B_{V_i,t-1}, b_{V_i,t}).$$

Thus the compressed cache grows with r_i rather than d_h per token, while the bases are stored once per layer and head. Our algorithm can be summarized as the pseudo-code in Appendix B. The storage and computational ratios are approximately r/d when the sequence length is large; Appendix I provides the detailed accounting.

Table 1: Average latency per attention layer for token generation. Both the compression rates for AURA-KV and Palu are 50%. AURA-KV + KV Comp. indicates using our method to retain half after compressing the KV size to 2048. Mistral-7B, Longchat-7B and Llama represent Mistral-7B, Longchat-7B and Llama-3-8B, respectively.

Model	Baseline	Palu	AURA-KV	AURA-KV+ KV Comp.
Mistral-7B	1.00×	1.30×	2.75×	**3.23**×
Longchat-7B	1.00×	1.02×	1.89×	**2.13**×
Llama3-8B	1.00×	1.03×	1.93×	**2.02**×

4 Experiments

4.1 Models and datasets

We evaluate AURA-KV on Llama-3-8B-Instruct (Dubey et al., 2024), Mistral-7B-Instruct (Jiang et al., 2023), and LongChat-7B (Li et al., 2023). Our main quality evaluation uses LongBench (Bai et al., 2024), which contains 17 tasks covering single-document QA (Kočiskỳ et al., 2018; Dasigi et al., 2021), multi-document QA (Yang et al., 2018; Ho et al., 2020), summarization (Huang et al., 2021; Zhong et al., 2021; Fabbri et al., 2019), few-shot learning (Joshi et al., 2017; Gliwa et al., 2019), synthetic reasoning, and code completion (Guo et al., 2023; Liu et al., 2023). Most LongBench inputs contain 5k–15k tokens, making the benchmark suitable for evaluating long-context cache compression. We report category averages and the overall average, where higher is better. We organize the existing experiments around three questions: whether feature-dimension compression reduces decoding cost, whether it preserves long-context quality compared with low-rank baselines, and whether it can be used as a second-stage compressor with token eviction or quantization.

4.2 Efficiency and quality on long-context benchmarks

Decoding efficiency. Table 1 reports average per-layer decoding latency. At the same 50% rank budget, AURA-KV is consistently faster than Palu, reaching 2.75×, 1.89×, and 1.93× speedups on Mistral-7B, LongChat-7B, and Llama-3-8B, respectively. The main implementation difference is that Palu applies a learned/estimated projection around the KV representation, whereas AURA-KV factorizes the already rotated cache and keeps decoding in the compressed post-RoPE space. This avoids full-dimensional reconstruction during decoding and lets the rank budget vary by layer and by key/value cache; Appendix E gives a side-by-side comparison. When AURA-KV is applied after token-level KV-size reduction, the speedup further increases to 3.23×, 2.13×, and 2.02×. This shows that the compressed representation reduces the cost of the decoding path, not only the cache footprint. Table 7 reports the one-time prefill overhead of the low-rank solver, and Appendix R gives the corresponding complexity discussion.

Long-context quality. Table 2 compares AURA-KV with Palu at 70% and 50% rank budgets. AURA-KV achieves stronger average accuracy in nearly all model/budget settings, with the largest gains appearing under more aggressive compression. For example, on Llama-3-8B at the 50% budget, Palu drops to 28.11 average while AURA-KV retains 38.55. Appendix L reports additional Palu results when it is combined with token-eviction methods. At the 70% budget, AURA-KV often matches the full-cache baseline and sometimes gives small positive deltas; we interpret these as robustness under moderate compression rather than as a primary improvement claim. In addition, Appendix K reports additional compression ratios and results on RULER and InfiniteBench. Appendix Q reports newer model versions and additional low-rank baselines, with baseline coverage summarized in Appendix Q.2. Thus the empirical suite covers feature-rank compression, token eviction, quantization, and two long-context stress benchmarks, although we still leave same-hardware reproduction of ThinK and LoRC to future work.

Overall, performance decreases as the retained rank falls from 70% to 50%, but the degradation is substantially smaller for AURA-KV than for Palu. This supports the two design choices in Section 3: compressing the post-RoPE cache directly and allocating rank adaptively across layers.

4.3 Compositional compression

We report the KV-cache memory after two-stage compression in Table 3 and combine AURA-KV with four token-eviction baselines: H2O (Zhang et al., 2023), StreamLLM (SLM) (Xiao et al., 2023b), SnapKV (SKV) (Li et al., 2024), and PyramidKV (PKV) (Cai et al., 2024). Table 4 shows the results on Llama-3-8B. Because these methods reduce the sequence dimension while AURA-

Table 2: Results on LongBench, with each column displaying average scores across tasks.

Model	Method	Comp. Ratio	Multi-QA	Single-QA	Summarization	Few-Shot	Code	Synthetic	Avg.
Mistral-7B	Baseline	1.00x	36.13	39.53	25.17	69.35	57.33	50.00	45.32
	Palu-70%	1.43x	38.36	38.88	**27.40**	67.45	56.51	47.20	45.23
	AURA-KV-70%		**38.37**	**39.82**	26.65	**69.32**	**60.91**	**50.00**	**46.52**
	Palu-50%	2.00x	33.64	**38.04**	23.83	67.47	54.77	33.5	41.59
	AURA-KV-50%		**34.24**	32.91	**24.43**	**69.41**	**57.37**	**34.02**	**41.61**
Longchat-7B	Baseline	1.00x	23.95	31.12	26.74	63.80	56.91	15.25	36.32
	Palu-70%	1.43x	22.42	29.43	25.52	62.87	**58.99**	14.25	35.45
	AURA-KV-70%		**24.24**	**30.96**	**26.64**	**63.31**	57.14	**15.00**	**36.29**
	Palu-50%	2.00x	22.61	25.33	**22.73**	60.12	43.52	6.84	30.82
	AURA-KV-50%		**23.50**	**26.28**	16.80	**61.57**	**49.38**	**10.21**	**31.48**
Llama-3-8B	Baseline	1.00x	34.59	32.19	24.95	68.48	54.41	36.96	41.46
	Palu-70%	1.43x	25.05	30.35	24.20	**69.69**	40.05	31.81	36.98
	AURA-KV-70%		**35.45**	**33.29**	**24.54**	66.94	**56.71**	**37.36**	**41.80**
	Palu-50%	2.00x	25.50	28.54	20.90	64.55	15.66	8.5	28.11
	AURA-KV-50%		**31.89**	**33.13**	**22.41**	**65.60**	**45.41**	**33.48**	**38.55**

Table 3: KV-cache memory for Llama-3-8B under sequence-length and rank-ratio compression. KV Size = 2048 applies a token-level cache budget before feature-dimension compression.

KV Size	Rank Ratio	Comp. Ratio	Memory
Full KV	100%	100%	6848M
	70%	70%	4793M
	50%	50%	3424M
KV Size = 2048	100%	25.0%	1712M
	70%	17.5%	1198M
	50%	12.5%	856M

KV reduces the feature dimension, they are naturally composable. After reducing KV size to 2048 tokens, applying AURA-KV at 70% rank usually preserves the original score within a small range, while 50% rank yields much larger memory reduction with more visible but controlled quality loss. Appendix M extends the compatibility study to Mistral-7B-Instruct, and Appendix N reports combinations with KV-cache quantization.

4.4 Ablations and prefill overhead

Adaptive rank allocation. The adaptive budget in Eq. 2 is intended to spend rank where the cached spectrum is less concentrated, instead of forcing every layer to use the same feature dimension. Table 5 compares AURA-KV with uniform compression (UC), where each layer receives the same compression ratio. The adaptive policy consistently improves LongBench averages, with larger gains at the more aggressive 50% budget: $+4.04$ on Mistral-7B, $+3.99$ on LongChat-7B, and $+4.59$ on Llama-3-8B. This ablation supports the use of layer-wise spectral allocation rather than treating the cache as equally compressible across depth. Table 6 further shows representative ranks selected by the adaptive policy. The selected ranks are not constant across layers or between keys and values, which is consistent with the layer-wise spectral variation observed in Fig. 2. For example, in the 50% budget, the selected key rank drops from 78 at Layer 11 to 45 at Layers 22 and 28, while value ranks follow a different pattern. This non-uniformity is useful in practice because it converts a single global memory target into layer-specific rank choices without hand-tuning a schedule for each model. Appendix P additionally plots the full key/value rank schedules across all layers.

Prefill overhead. AURA-KV computes low-rank bases once after the prompt prefill and then reuses them throughout decoding. The extra prefill cost is therefore paid once per request, while the latency benefit in Table 1 applies at every decoding step. Table 7 shows that the randomized low-rank solver reduces prefill throughput by about 45% on Llama-3-8B-Instruct, which is small relative to the repeated decoding-time savings in long-generation settings. This separation between one-time prefill work and repeated decode acceleration

Table 4: Performance comparison of our method combined with PyramidKV (PKV), SnapKV (SKV), H2O, StreamingLLM (SLM) (KV Size = 2048), and FullKV (FKV) on LongBench for Llama-3-8B. R& Q stands for Low rank compression and Quantization. AURA KV, combined with KV size reduction methods, significantly reduces storage requirements with minimal performance degradation.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
	18409	3619	4559	9151	4887	11214	8734	10614	2113	5177	8209	6258	11141	9289	1235	4206	
Llama-3-8B-Instruct, Full KV																	
100%Rank	25.70	29.75	**41.12**	45.55	35.87	22.35	**25.63**	23.03	**26.21**	**73.0**	**90.56**	**41.88**	4.67	**69.25**	58.05	50.77	41.46
70%Rank	**25.24**	**35.21**	39.41	**46.22**	**37.7**	**22.42**	24.22	**23.22**	26.18	71.5	90.07	39.24	5.71	69.00	**59.69**	**53.72**	**41.80**
50%Rank	24.52	36.16	38.72	43.72	31.09	20.85	18.97	22.47	25.78	72.5	87.94	36.36	5.76	61.19	47.16	43.66	38.55
50%R&Q	23.31	32.92	39.42	43.27	32.72	20.99	17.93	21.0	22.48	70.5	87.39	36.64	**6.10**	68.75	49.32	41.11	38.36
Llama-3-8B-Instruct, Pyramid KV, KV Size=2048																	
100%Rank	25.40	29.71	**40.25**	44.76	35.32	21.98	**26.83**	**23.30**	26.19	**73.0**	**90.56**	**42.14**	5.22	69.25	58.76	51.18	41.49
70%Rank	**25.91**	**34.62**	39.86	**45.73**	**40.82**	**22.61**	26.18	23.20	**26.65**	71.5	90.20	40.70	**5.73**	**69.25**	**59.85**	**53.46**	**42.27**
50%Rank	23.68	33.56	34.92	42.26	34.23	21.03	23.03	22.41	25.56	72.0	87.73	37.35	5.33	68.79	56.10	49.91	39.87
50%R&Q	23.29	33.31	33.87	41.38	35.12	20.34	22.73	21.95	25.41	72.0	87.60	35.42	5.02	68.67	55.50	48.34	39.37
Llama-3-8B-Instruct, SnapKV, KV Size=2048																	
100%Rank	**25.86**	29.55	**41.10**	44.99	35.80	**21.81**	**25.98**	**23.40**	**26.46**	**73.5**	**90.56**	**41.66**	5.17	**69.25**	56.65	49.94	**41.35**
70%Rank	25.72	**33.14**	37.96	**45.41**	**37.15**	20.08	22.88	22.17	25.49	70.0	89.43	39.40	**6.12**	69.00	**57.16**	**50.87**	40.75
50%Rank	24.75	27.41	38.07	44.95	34.03	21.39	15.21	21.03	20.25	68.0	88.39	36.19	6.00	69.00	44.96	38.48	37.38
50%R&Q	24.07	27.42	39.26	43.71	33.63	21.07	15.51	20.53	20.76	68.0	88.75	34.32	5.58	68.36	44.55	37.45	37.06
Llama-3-8B-Instruct, H2O, KV Size=2048																	
100%Rank	**25.56**	26.85	39.54	44.30	**32.92**	**21.09**	24.68	**23.01**	**26.16**	53.0	**90.56**	**41.84**	**4.91**	69.25	**56.40**	49.68	39.35
70%Rank	24.52	37.86	**40.59**	**44.83**	29.50	19.72	**26.06**	22.53	26.00	**56.5**	88.72	40.71	4.34	**69.63**	55.05	**50.73**	**39.83**
50%Rank	22.34	29.56	37.01	36.99	29.40	18.68	18.11	21.02	21.62	52.5	88.95	40.41	4.10	68.75	48.36	35.99	35.86
50%R&Q	23.08	29.61	37.46	36.01	29.60	19.93	17.81	20.18	21.11	52.5	87.83	38.15	4.00	69.0	45.46	35.72	35.46
Llama-3-8B-Instruct, StreamingLLM, KV Size=2048																	
100%Rank	21.71	25.78	38.13	40.12	32.01	**16.86**	23.14	**22.64**	26.48	70.0	83.22	31.75	5.74	68.50	**53.50**	**45.58**	37.82
70%Rank	**21.78**	26.10	**38.48**	**40.60**	**32.96**	16.23	**23.37**	21.37	**26.48**	68.0	85.42	35.19	**6.00**	**69.00**	53.07	45.22	**38.08**
50%Rank	22.33	26.29	33.13	26.90	22.87	14.13	24.40	21.82	24.93	71.0	87.18	37.19	5.88	48.21	48.13	44.41	34.92
50%R&Q	23.34	**26.42**	33.06	25.93	18.46	12.37	25.02	21.66	25.58	**71.5**	**88.16**	**38.00**	5.57	42.44	48.67	43.61	34.36

Table 5: Ablation of adaptive rank allocation on LongBench. UC denotes uniform compression with the same rank ratio in every layer.

Method	Comp.	Mistral	LongChat	Llama
Baseline	1.00x	45.32	36.32	41.46
AURA-KV-70%	1.43x	46.52	36.29	41.80
UC-70%	1.43x	44.57	35.57	41.57
AURA-KV-50%	2.00x	37.61	31.27	38.55
UC-50%	2.00x	33.57	27.28	33.96

Table 6: Representative layer-wise ranks selected by AURA-KV on the Llama-3-8B-Instruct setting. We report Key/Value ranks as "K / V".

Layer	5	11	17	22	28	31
70% (K / V)	82 / 119	94 / 122	89 / 114	68 / 110	68 / 102	65 / 108
50% (K / V)	70 / 93	78 / 87	73 / 81	45 / 68	45 / 61	53 / 69

is important for serving: AURA-KV is most attractive when the generation length or batch concurrency makes KV-cache bandwidth the bottleneck. The reported compressed prefill throughput includes both key and value dimensionality reduction for the entire prefill cache, so it directly measures the operation added after prefill rather than only the subsequent decode path.

Table 7: Prefill throughput on `Llama-3-8B-Instruct` (A40, tokens/s). The compressed setting includes the one-time LowRankSolver cost.

Ctx.	Standard Prefill	w/ AURA-KV Compr.	Drop
4k	232.59	220.96	∼5.0%
16k	242.31	232.62	∼4.0%

Fixed-basis decoding. All reported results use the fixed bases computed after prefill; Appendix T discusses this assumption and its limitations.

5 Conclusion

We presented AURA-KV, a training-free low-rank approximation method for efficient long-context LLM inference. By compressing the post-RoPE KV cache and allocating layer-wise ranks under a global memory budget, AURA-KV reduces cache memory and decoding cost while preserving long-context quality. The method is complementary to token eviction and quantization, making it a practical second-stage compression module.

6 Limitations

Our theoretical analysis is intentionally idealized: it studies a simplified attention layer and does not fully model LayerNorm, residual connections, MLP blocks, or trained-weight dependencies. The Gaussian-weight assumption is used only as motivation, so the proposition should be interpreted as an explanatory lens rather than a necessary condition for AURA-KV. The main evidence for practical effectiveness comes from empirical rank allocation and ablations.

Since AURA-KV compresses the head dimension, its stand-alone compression ratio is bounded by the per-head dimension, e.g., $d_h = 128$ in our main models; larger reductions require combination with token pruning or quantization. The current implementation keeps the prefill basis fixed during decoding, and periodic basis refresh for very long generations remains untested. Aggressive rank budgets may degrade accuracy, while small positive deltas from compression are not fully explained.

Our controlled comparisons cover Palu and Eigen Attention, while reproducing ThinK and LoRC under identical kernels and hardware is left to future work. We have not evaluated Qwen-family models, and our newer-model evidence is limited to Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. Although we include several long-context benchmarks, we do not provide dedicated multilingual, code-generation, safety, or bias evaluations.

7 Ethical Statement

AURA-KV is a training-free KV-cache compression method intended to improve inference efficiency, not safety or alignment. We use public models and benchmarks, do not collect personal data, and do not redistribute third-party weights or datasets.

Lower memory and latency may benefit long-context applications, but can also reduce the cost of misuse. The method does not address toxicity, misinformation, privacy leakage, or unsafe generation, so deployments should include content filtering, abuse monitoring, rate limiting, human oversight, and task-specific audits. Compression may also introduce input-dependent deviations across languages, tasks, or groups; our benchmarks do not certify fairness or safety. Environmentally, AURA-KV may reduce decoding-time compute and memory, but actual savings depend on hardware and workload.

References

Anonymous Author. ReasonAlloc: Hierarchical decoding-time KV cache budget allocation for reasoning models. Under double-blind review.

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. *arXiv preprint arXiv:2308.14508*. ACL 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Chi-Chih Chang, Chien-Yu Lin, Yash Akhauri, Wei-Cheng Lin, Kai-Chiang Wu, Luis Ceze, and Mohamed S Abdelfattah. 2025. xkv: Cross-layer svd for kv-cache compression. *arXiv preprint arXiv:2503.18893*.

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. 2024. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Yao Fu. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv preprint arXiv:2405.08944*.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR.

Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P Woodruff, and Amir Zandieh. 2023. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Preprint*, arXiv:2401.18079.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. 2023. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(8):10173–10196.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *Preprint*, arXiv:2403.05527.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

P. Lévy. 1925. *Calcul des probabilités, par Paul Lévy, ...* Gauthier-Villars.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

Wenjuan Li, Bing Li, Chunfeng Yuan, Yangxi Li, Haohao Wu, Weiming Hu, and Fangshi Wang. 2020. Anisotropic convolution for image classification. *IEEE Transactions on Image Processing*, 29:5584–5595.

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. *arXiv preprint arXiv:2404.14469*.

Weixin Liang, Yuhui Zhang, Yongchan Kwon, Serena Yeung, and James Y. Zou. 2022. Mind the gap: Understanding the modality gap in multi-modal contrastive representation learning. *ArXiv*, abs/2203.02053.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *URL https://arxiv.org/abs/2306.03091*.

Yijiang Liu, Huanrui Yang, Zhen Dong, Kurt Keutzer, Li Du, and Shanghang Zhang. 2022. Noisyquant: Noisy bias-enhanced post-training activation quantization for vision transformers. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20321–20330.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization.

Victor M. Panaretos and Yoav Zemel. 2018. Statistical aspects of wasserstein distances. *Annual Review of Statistics and Its Application*.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. 2016. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

John G Saw, Mark CK Yang, and Tse Chin Mo. 1984. Chebyshev inequality with estimated mean and variance. *The American Statistician*, 38(2):130–132.

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, Miami, Florida, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023a. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023b. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2024. ThinK: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*. ICLR 2025 Spotlight (arXiv v1: 2024-07-30).

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. 2024. LoRC: Low-rank compression for LLMs KV cache with a progressive compression strategy. *arXiv preprint arXiv:2410.03111*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and 1 others. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2023. Kivi : Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization.

A Proof of Proposition 2.1

Step 1. $l+1-$th layer $i-$th token can be shown as

$$x_i^{(l+1)T} = \text{softmax}\left(x_i^{(l)T} W_q^T R_i^T RoPE(W_k X_{1:i}^{(l)})\right) X_{1:i}^{(l)T} W_v^T, \tag{5}$$

where $x_i^{(l+1)}, x_i^{(l)}, X_{1:i}^{(l)}, R_i, \text{RoPE}(\cdot)$, means $i-$th token in $l+1$-th layer, $i-$th token in l-th layer, $[x_1^{(l)}, x_2^{(l)}, \ldots, x_i^{(l)}] \in \mathcal{R}^{d\times i}$, i-th RoPE matrix, RoPE transformation, respectively. For convenience, we denote

$$\mathbf{s}_{1:i} = \left[\exp\left(x_i^{(l)T} W_q^T R_i^T R_{\hat{i}} W_k x_{\hat{i}}^{(l)}\right)\right]_{\hat{i}\leq i} \in \mathcal{R}^i \tag{6}$$

Thus, $x_i^{(l+1)}$ can be written as

$$x_i^{(l+1)T} = W_v X_{1:i}^{(l)} \mathbf{s}_{1:i}, \tag{7}$$

where $\mathbf{s}_{1:i}$ is tensor slicing. And we have

$$\begin{aligned} E(\|x_i^{(l+1)}\|^2) &= E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} W_v^T W_v X_{1:i}^{(l)} \mathbf{s}_{1:i}\right) \\ &= E\left(tr(\mathbf{s}_{1:i}\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} W_v^T W_v X_{1:i}^{(l)})\right) \\ &= tr\left(E(\mathbf{s}_{1:i}\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} W_v^T W_v X_{1:i}^{(l)})\right) \\ &= tr\left(E(\mathbf{s}_{1:i}\mathbf{s}_{1:i}^T) X_{1:i}^{(l)T} E(W_v^T W_v) X_{1:i}^{(l)}\right), \end{aligned} \tag{8}$$

Since $W_{v,n,m} \sim \mathcal{N}(0, 1/d^2)$, we have

$$E(W_v^T W_v) = \frac{1}{d}\mathbf{I}, \tag{9}$$

where $\mathbf{I}$ is identity matrix. Thus, $E(\|x_i^{(l+1)}\|^2)$ can be written as

$$E(\|x_i^{(l+1)}\|^2) = \frac{1}{d} E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right). \tag{10}$$

Besides, we have

$$E(\|x_i^{(l+1)}\|^4) = \frac{3}{d^4} E\left((\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i})^2\right) \leq \frac{3}{d^4} E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)^2 \tag{11}$$

Thus, according to chebyshev's inequality (Saw et al., 1984), we have

$$\begin{aligned} P\left\{\left|\|x_i^{(l+1)}\|^2 - E(\|x_i^{(l+1)}\|^2)\right| \geq \epsilon_1\right\} &= P\left\{\left|\|x_i^{(l+1)}\| - \frac{1}{d} E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)\right| \geq \epsilon_1\right\} \\ &\leq \frac{12 E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)^2}{d^4 \epsilon_1^2} \end{aligned} \tag{12}$$

Thus,

$$P\left\{\left|\frac{d\|x_i^{(l+1)}\|^2}{E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)} - 1\right| \geq \epsilon_1\right\} \leq \frac{12}{d^2\epsilon_1^2} = O\left(\frac{1}{d^2\epsilon_1^2}\right) \tag{13}$$

Thus, with probability at least $1 - O(1/(d^2\epsilon_1^2))$, we have

$$\left| \frac{d\|x_i^{(l+1)}\|^2}{E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)} - 1 \right| < \epsilon_1, \tag{14}$$

which implies that with probability at least $1 - O(1/(d^2\epsilon^2))$ the following holds,

$$\frac{1}{\|x_i^{(l+1)}\|} > \left(\frac{d}{E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right)} \right)^{\frac{1}{2}} \left(1 - \frac{\epsilon_1}{2}\right). \tag{15}$$

Similarly, since

$$\cos\langle x_i^{(l+1)}, x_j^{(l+1)} \rangle = \cos\langle W_v X_{1:i}^{(l)} \mathbf{s}_{1:i}, W_v X_{1:j}^{(l)} \mathbf{s}_{1:j} \rangle \tag{16}$$

we obtain the following similar result, for any $\epsilon_2 > 0$, with probability at least $1 - O(1/(d^2\epsilon_2^2))$

$$\left| \frac{x_i^{(l+1)T} x_j^{(l+1)}}{dE\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right)} - 1 \right| < \epsilon_2, \tag{17}$$

which implies

$$x_i^{(l+1)T} x_j^{(l+1)} > \frac{E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right)}{d}(1 - \epsilon_2), \quad \text{if} \quad E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right) > 0, \tag{18}$$

$$x_i^{(l+1)T} x_j^{(l+1)} < \frac{E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right)}{d}(1 - \epsilon_2), \quad \text{if} \quad E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right) < 0. \tag{19}$$

Thus, we have

$$\left| x_i^{(l+1)T} x_j^{(l+1)} \right| > \frac{\left| E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right) \right|}{d}(1 - \epsilon_2). \tag{20}$$

Combining (15) and (18), with probability at least $1 - O(1/(d^2\epsilon_1^2)) - O(1/(d^2\epsilon_2^2))$, we have

$$\begin{aligned} |\cos\langle x_i^{(l+1)}, x_j^{(l+1)} \rangle| &= \frac{|x_i^{(l+1)T} x_j^{(l+1)}|}{\|x_i^{(l+1)}\| \|x_j^{(l+1)}\|} \\ &> \frac{\left| E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right) \right|}{\left(E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{s}_{1:i}\right) E\left(\mathbf{s}_{1:j}^T X_{1:j}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right)\right)^{\frac{1}{2}}} \left(1 - \frac{\epsilon_1}{2}\right)^2 (1 - \epsilon_2). \end{aligned} \tag{21}$$

step2: Similarly,

$$E\left(\mathbf{s}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{s}_{1:j}\right) = tr\left(X_{1:i}^{(l)T} X_{1:j}^{(l)} E\left(\mathbf{s}_{1:j} \mathbf{s}_{1:i}^T\right)\right), \tag{22}$$

and we have

$$\begin{aligned} E\left(\mathbf{s}_{1:i} \mathbf{s}_{1:j}^T\right)_{\hat{i},\hat{j}} &= E\left(\exp\left(x_i^{(l)T} W_q^T R_i^T R_{\hat{i}} W_k x_{\hat{i}}^{(l)} + x_j^{(l)T} W_q^T R_j^T R_{\hat{j}} W_k x_{\hat{j}}^{(l)}\right)\right) \\ &= E\left(\exp\left(tr\left(x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T R_i^T R_{\hat{i}} W_k + x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T R_j^T R_{\hat{j}} W_k\right)\right)\right) \\ &= E\left(\exp\left(tr\left(\left(x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T R_i^T R_{\hat{i}} + x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T R_j^T R_{\hat{j}}\right) W_k\right)\right)\right). \end{aligned} \tag{23}$$

For convenience, we denote

$$A_{\hat{i},\hat{j}} = x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T R_i^T R_{\hat{i}} + x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T R_j^T R_{\hat{j}}. \tag{24}$$

Due to W_k is normal matrix random variable, according to eigen function(Lévy, 1925) we have

$$E\left(\mathbf{s}_{1:i}\mathbf{s}_{1:j}^T\right)_{\hat{i},\hat{j}} = E\left(\exp\left(\frac{1}{2d} tr(A_{\hat{i},\hat{j}} A_{\hat{i},\hat{j}}^T)\right)\right). \tag{25}$$

Furthermore, we have

$$\begin{aligned} tr(A_{\hat{i},\hat{j}} A_{\hat{i},\hat{j}}^T) = \quad & tr(x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T R_i^T R_{\hat{i}}^T R_{\hat{j}} R_j W_q x_j^{(l)} x_{\hat{j}}^{(l)T} \\ & + x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T W_q x_i^{(l)} x_{\hat{i}}^{(l)T} \\ & + x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T W_q x_i^{(l)} x_{\hat{j}}^{(l)T} \\ & + x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T R_j^T R_{\hat{j}}^T R_{\hat{i}} R_i W_q x_i^{(l)} x_{\hat{i}}^{(l)T}) \\ = \quad & tr(x_{\hat{j}}^{(l)T} x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T R_i^T R_{\hat{i}}^T R_{\hat{j}} R_j W_q x_j^{(l)} \\ & + x_{\hat{i}}^{(l)T} x_{\hat{i}}^{(l)} x_i^{(l)T} W_q^T W_q x_i^{(l)} \\ & + x_{\hat{j}}^{(l)T} x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T W_q x_i^{(l)} \\ & + x_{\hat{i}}^{(l)T} x_{\hat{j}}^{(l)} x_j^{(l)T} W_q^T R_j^T R_{\hat{j}}^T R_{\hat{i}} R_i W_q x_i^{(l)}). \end{aligned} \tag{26}$$

Due to $x_{\hat{i}}^{(l)T} x_{\hat{i}}^{(l)} = x_{\hat{j}}^{(l)T} x_{\hat{j}}^{(l)} = 1$, we have

$$-2x_j^{(l)T} W_q^T W_q x_j^{(l)} - 2x_i^{(l)T} W_q^T W_q x_i^{(l)} \leq tr(A_{\hat{i},\hat{j}} A_{\hat{i},\hat{j}}^T) \leq 2x_j^{(l)T} W_q^T W_q x_j^{(l)} + 2x_i^{(l)T} W_q^T W_q x_i^{(l)} \tag{27}$$

Furthermore, we have

$$E\left(\exp\left(\frac{2}{d} x_j^{(l)T} W_q^T W_q x_j^{(l)}\right)\right) = E\left(\exp\left(\frac{2}{d} x_i^{(l)T} W_q^T W_q x_i^{(l)}\right)\right) = \left(1 - \frac{4}{d^2}\right)^{-d/2}, \tag{28}$$

$$E\left(\exp - \left(\frac{2}{d} x_j^{(l)T} W_q^T W_q x_j^{(l)}\right)\right) = E\left(\exp - \left(\frac{2}{d} x_i^{(l)T} W_q^T W_q x_i^{(l)}\right)\right) = \left(1 + \frac{4}{d^2}\right)^{-d/2}. \tag{29}$$

Thus, we have

$$\left(1 + \frac{4}{d^2}\right)^{-d/2} \leq E\left(\mathbf{s}_{1:i}^T \mathbf{s}_{1:i}\right)_{\hat{i},\hat{j}} \leq \left(1 - \frac{4}{d^2}\right)^{-d/2}. \tag{30}$$

It is worth noting that

$$\lim_{d \to +\infty} \left(1 - \frac{4}{d^2}\right)^{-d/2} = \lim_{d \to +\infty} \left(1 + \frac{4}{d^2}\right)^{-d/2} = 1. \tag{31}$$

Thus, we have

$$\begin{aligned} |\cos\langle x_i^{(l+1)}, x_j^{(l+1)}\rangle| > & \left(\frac{\left|\mathbf{1}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{1}_{1:j}\right|}{\left(\mathbf{1}_{1:i}^T X_{1:i}^{(l)T} X_{1:i}^{(l)} \mathbf{1}_{1:i}\right)^{\frac{1}{2}} \left(\mathbf{1}_{1:j}^T X_{1:j}^{(l)T} X_{1:j}^{(l)} \mathbf{1}_{1:j}\right)^{\frac{1}{2}}} - |\Delta|\right) \left(1 - \frac{\epsilon_1}{2}\right)^2 (1 - \epsilon_2), \\ = & \left(\frac{\left|\mathbf{1}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{1}_{1:j}\right|}{\left\|X_{1:i}^{(l)} \mathbf{1}_{1:i}\right\| \left\|X_{1:j}^{(l)} \mathbf{1}_{1:j}\right\|} - |\Delta|\right) \left(1 - \frac{\epsilon_1}{2}\right)^2 (1 - \epsilon_2), \end{aligned} \tag{32}$$

where $\Delta \in \mathcal{O}(1/d)$ is a minor perturbation.

We notice that

$$\frac{\left|\mathbf{1}_{1:i}^T X_{1:i}^{(l)T} X_{1:j}^{(l)} \mathbf{1}_{1:j}\right|}{\left\|X_{1:i}^{(l)}\mathbf{1}_{1:i}\right\| \left\|X_{1:j}^{(l)}\mathbf{1}_{1:j}\right\|} = |\cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle|, \tag{33}$$

and we denote $\epsilon_1^2/4 - \epsilon_1 - \epsilon_2 - \epsilon_1\epsilon_2 - \epsilon_1^2\epsilon_2/4 = -\epsilon$,thus, eqn 32 can be written as

$$\begin{aligned} |\cos\langle x_i^{(l+1)}, x_j^{(l+1)}\rangle| &> \left(|\cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle| - |\Delta|\right)(1-\epsilon), \\ &> \cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle| - |\Delta| - \epsilon\left(|\cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle| - |\Delta|\right). \end{aligned} \tag{34}$$

Therefore, we choose ϵ_1, ϵ_2 such that $\epsilon \left||\cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle| - |\Delta|\right| \epsilon < |\Delta|$, we have

$$|\cos\langle x_i^{(l+1)}, x_j^{(l+1)}\rangle| > \cos\langle \bar{x}_i^{(l)}, \bar{x}_j^{(l)}\rangle| - 2|\Delta|, \tag{35}$$

where $2|\Delta| \in \mathcal{O}(1/d)$ and complete the proof.

B Algorithm Framework

Algorithm 1 AURA-KV

Input: Initial caches $\{K_{i,0}, V_{i,0}\}_{i=1}^h$, prefill length T, target key/value ranks $\{r_i^K, r_i^V\}_{i=1}^h$, buffer $\mathcal{B}$, head dimension d_h.
for $t = 1, \ldots, n$ **do**
  $q_t = \text{concatenate}(q_{1,t}, q_{2,t}, \ldots, q_{h,t})$
  for *each head i in parallel* **do**
    if $t < T$ **then**
      Append $k_{i,t}$ and $v_{i,t}$ to the full caches $K_{i,t-1}$ and $V_{i,t-1}$;
      $P_i = q_{i,t}K_{i,t}^T$; $O_i = \text{softmax}(P_i/\sqrt{d_h})V_{i,t}$
    else if $t == T$ **then**
      Append $k_{i,t}$ and $v_{i,t}$ to the full caches;
      $P_i = q_{i,t}K_{i,t}^T$; $O_i = \text{softmax}(P_i/\sqrt{d_h})V_{i,t}$
      $A_{K_i}, B_{K_i} = \textsc{LowRankSolver}(K_i^T, r_i^K)$;
      $A_{V_i}, B_{V_i} = \textsc{LowRankSolver}(V_i^T, r_i^V)$;
      Replace K_i, V_i in $\mathcal{B}$ with $A_{K_i}, B_{K_i}, A_{V_i}, B_{V_i}$;
    else
      Append $A_{K_i}^T k_{i,t}$ and $A_{V_i}^T v_{i,t}$ to B_{K_i} and B_{V_i};
      $P_i = (q_{i,t}A_{K_i})B_{K_i}$;
      $O_i = [\text{softmax}(P_i/\sqrt{d_h})B_{V_i}^T]A_{V_i}^T$.
    end if
  end for
  $O = \textbf{concatenate}(O_1, O_2, \ldots, O_h)$.
end for

C Related work

KV Cache Compression. KV-cache compression methods can be grouped by what they remove or approximate. Token eviction methods reduce the sequence dimension by retaining important cached positions: H2O keeps heavy-hitter tokens (Zhang et al., 2023), StreamLLM preserves attention-sink tokens for streaming generation (Xiao et al., 2023b), SnapKV selects head-specific prompt positions (Li et al., 2024), and PyramidKV allocates different cache budgets across layers (Cai et al., 2024). Quantization methods instead reduce the precision of cached activations, as in KVQuant, KIVI, and GEAR (Hooper et al., 2024; Liu et al., 2024; Kang et al., 2024). These approaches are effective but largely orthogonal to feature-dimension compression.

Low-rank KV-cache methods compress the feature dimension of keys and values. Palu projects cached states into a lower-dimensional space (Chang et al., 2024), while Eigen Attention performs attention in a low-rank subspace (Saxena et al., 2024). LoRC and ThinK further explore progressive low-rank compression and query-driven channel pruning (Zhang et al., 2024; Xu et al., 2024). More recently, xKV exploits cross-layer redundancy by extracting aligned singular-vector subspaces from grouped layers, thereby sharing low-rank representations across layers rather than compressing each layer independently (Chang et al., 2025). AURA-KV differs in two respects. First, it compresses the post-RoPE cache directly, avoiding repeated full-dimensional reconstruction during decoding. Second, it uses adaptive layer-wise rank allocation under a global memory budget, rather than relying on a fixed layer grouping or a uniform rank across layers. Appendix E gives a more detailed complexity comparison.

Moreover, recent parallel work explores hierarchical decoding-time KV-cache budget allocation for reasoning models (Author).

Cone Effect. The cone effect describes anisotropic clustering of high-dimensional representations. It has been studied in neural networks and representation learning, where normalization, architectural bias, and contractive mappings can shape the angular distribution of hidden states (Huang et al., 2023; Li et al., 2020; Liu et al., 2022; Liang et al., 2022). We use this line of work to motivate why deeper Transformer representations may become easier to approximate with fewer spectral components. Our analysis is intentionally idealized, and our empirical rank-allocation ablations serve as the primary validation that this motivation is useful for KV-cache compression.

D Assumption Demonstration on Longchat

As illustrated in Fig. 5, the Wasserstein distance between the empirical coefficient distribution W_q, W_k, W_v and the reference normal distribution $\mathcal{N}(0, d^{-2})$ for each layer of LongChat remains on the order of 10^{-3}. This close marginal alignment provides additional support for using the Gaussian model as an idealized approximation across multiple model architectures, but it does not establish that the assumption holds universally or for all normalization and initialization choices.

E Comparison with Eigen Attention and Palu

This appendix clarifies the difference between AURA-KV and the closest low-rank baselines. Palu is the most direct empirical baseline in our experiments because it also targets feature-dimension compression of the KV cache. The differences are summarized in Table 8.

Eigen Attention and Palu project keys into a lower-dimensional space before or around positional processing. Let $\hat{K} = U_k K$ denote the low-dimensional key cache. When RoPE must be applied in the original feature dimension, these methods require a reconstruction or remapping step during inference. The attention score can be written schematically as

$$q_i^T K = x_i^T W_q^T R_i^T RoPE(U_k^T \hat{K}), \tag{36}$$

where the computational complexity is $\mathcal{O}(d^2 + d + sr^2 + ds)$. AURA-KV instead performs the low-rank approximation directly on the post-RoPE key cache. The corresponding attention score is

$$q_i^T K = x_i^T W_q^T R_i A_K B_K, \tag{37}$$

where the computational complexity is $\mathcal{O}(d^2 + d + dr + rs)$. For long contexts, the sequence length s is the dominant factor, so avoiding the sr^2 and ds reconstruction-related terms reduces decoding overhead. This is the main source of AURA-KV's latency advantage in Table 1.

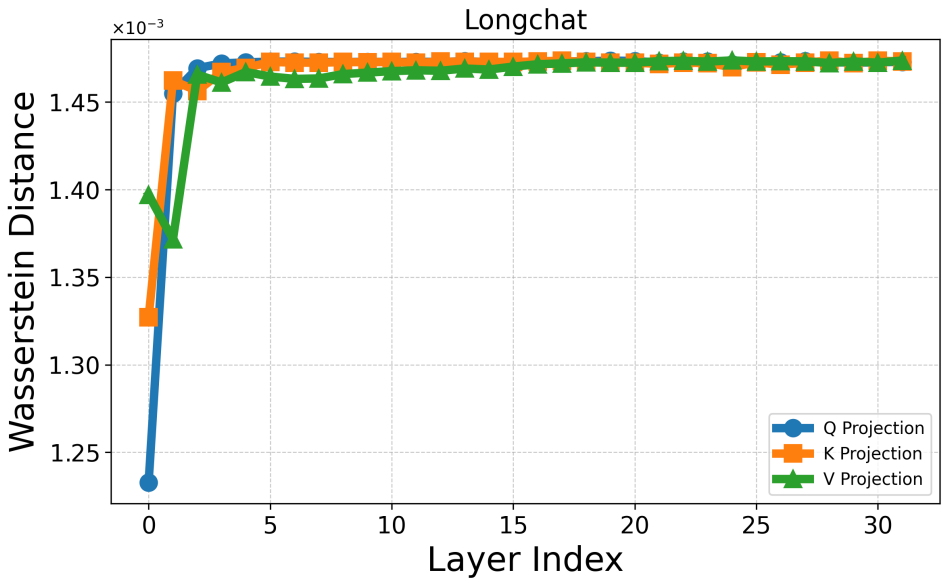


Figure 5: The Wasserstein distance between Longchat weights of W_q, W_k, W_v and $\mathcal{N}(0, d^{-2})$

Table 8: Key implementation and modeling differences between AURA-KV and Palu/Eigen-style low-rank KV-cache compression.

Aspect	**Palu / Eigen-style low-rank attention**	**AURA-KV**
Compression target	Projects cached representations into a reduced space before or around positional processing.	Factorizes the post-RoPE key and value caches that are actually used during decoding.
RoPE handling	May require reconstruction or remapping to apply RoPE in the original feature dimension.	Avoids reconstructing full-dimensional rotated caches after prefill; attention is computed directly with A_K, B_K, A_V, B_V.
Rank policy	Uses a fixed retained dimension or a less explicit global rank schedule.	Selects separate key/value ranks per layer through ADAPTIVERANKSEARCH under a global cache budget.
Decode update	Stores and updates compressed states according to the baseline projection rule.	Keeps the learned prefill bases fixed and appends only coefficients $A_{K_i}^\top k_{i,t}$ and $A_{V_i}^\top v_{i,t}$ for new tokens.
Empirical role	Serves as a direct low-rank baseline under matched rank budgets.	Improves latency and LongBench accuracy under the same 70% and 50% budgets in Tables 1 and 2.

F Relation to channel pruning and progressive low-rank compression.

ThinK (Xu et al., 2024) focuses on query-driven channel pruning, selecting a subset of original feature channels according to query interactions. AURA-KV instead uses subspace projection: it replaces the original coordinate axes with a low-rank basis estimated from the cached states. This distinction matters when features are correlated, because preserving principal directions can retain information that is spread across many original channels. LoRC (Zhang et al., 2024) also studies low-rank cache compression, but AURA-KV differs in its post-RoPE factorization and in its explicit global rank-budget allocation. These differences are reflected in the implementation goal: keep the decode path in the compressed space while making the rank schedule data-driven and layer-dependent.

G KV Compression Methods Compatible with AURA-KV

We primarily compare our approach with four mainstream KV cache algorithms. More importantly, our algorithm can be integrated with these four algorithms to jointly improve the compression rate.

StreamLLM (SLM). StreamingLLM is an efficient framework that leverages the attention sink phenomenon to improve the generalization of LLMs to longer texts. It allows LLMs trained with a finite length attention window to process infinite

sequence lengths without any fine-tuning. The key idea is to cache the key and value states (KV) of initial tokens, which are found to be the attention sinks, and use them during the decoding process. This approach reduces memory consumption and improves model performance in streaming applications.

Heavy Hitter Oracle (H2O). The H_2O algorithm identifies a small portion of tokens, called Heavy Hitters (H_2), which contribute most of the value when computing attention scores in Large Language Models (LLMs). By removing these H_2, the algorithm significantly reduces the memory footprint of the KV cache, improving the efficiency of LLMs. The authors demonstrate that the emergence of H_2 is natural and strongly correlates with the frequent co-occurrence of tokens in the text.

SnapKV (SKV). SnapKV is an innovative approach that optimizes the performance of Large Language Models (LLMs) by efficiently minimizing the Key-Value (KV) cache size. The approach leverages the consistent focus of attention heads on specific prompt attention features during generation, which can be obtained from an 'observation' window at the end of prompts. SnapKV automatically compresses KV caches by selecting clustered important KV positions for each attention head, reducing the computational overhead and memory footprint when processing long input sequences.

PyramidKV (PKV). The PyramidKV algorithm is a dynamic KV cache compression method that leverages the observed Pyramidal Information Funneling pattern in large language models. The algorithm adjusts the KV cache size across different layers, allocating more cache in lower layers and less in higher ones. This approach significantly reduces memory usage while maintaining performance comparable to models with a full KV cache. The algorithm can be integrated into existing large language models to optimize their memory usage and improve efficiency.

Full KV (FKV). KV cache stores all previous keys and values until the cache is full and then evicted.

H SVD Decomposition

SVD is a mathematical technique that decomposes a matrix into three distinct components, facilitating significant reductions in dimensionality and storage, and is especially useful in the context of large-scale data processing. For a given matrix K, SVD allows it to be expressed as the product of three matrices: U, Σ, and C^T [1], where U and C are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of K.

We focus on compressing the matrix to save storage and computation, we approximate K using a truncated SVD. This involves reducing K to its most significant singular values and corresponding vectors. Specifically, the matrix K can be approximated as $K \approx A_K B_K$, where $A_K = U_r$ and $B_K = \Sigma_r C_r^T$. Here, Σ_r contains only the r-largest singular values, and U_r and C_r are the corresponding left and right singular vectors. We can consider the matrix A_K as a mapping from a high-dimensional space to an r-dimensional space, while B_K represents an approximate representation of the original matrix K in this r-dimensional space. Similarly, we can perform the same operation for the matrix V.

I Storage and Computation Analysis

In this section, we will conduct a theoretical analysis for our algorithm using multi-head attention, which is the most commonly used in LLM. We first present the storage and computation accounting for multi-head attention. We then connect the idealized cone-effect proposition to the empirical motivation for adaptive rank allocation.

Storage Overhead. In the standard KV cache algorithm, assuming there are h heads in the multi-head attention mechanism, the primary distinction between multi-head and single-head attention lies in how the hidden dimensions of the K and V matrices are divided. Specifically, in multi-head attention, the hidden dimensions of both the K and V matrices are evenly distributed across h heads, with each head's matrix having a shape of $\frac{d}{h} \times s$. In contrast, our algorithm employs low-rank matrices A_K, B_K, A_V, and B_V to approximate the original K and V matrices. Denote $\Gamma_{storage}$ as the storage ratio of standard KV cache to our algorithm, we can get

$$\Gamma_{storage} = \frac{(2s\frac{r}{h} + 2\frac{dr}{h^2})h}{2sh\frac{d}{h}} = \frac{r}{sh} + \frac{r}{d} \approx \frac{r}{d}.$$

In practical applications, s is significantly greater

[1] Typically, the matrices in SVD are denoted by U, Σ, and V. However, in this paper, the Value Matrix is also represented by V. To avoid confusion, we use C to represent the V matrix in the SVD.

Table 9: Comparison with Eigen Attention (Saxena et al., 2024) under the same rank budgets on LongBench. Higher is better.

Model	Base	AURA-KV-70%	Eigen-70%	AURA-KV-50%	Eigen-50%
Llama-3-8B	41.46	**41.80**	37.21	**38.55**	31.71
Mistral-7B	45.32	**46.52**	43.12	**41.61**	36.16
LongChat-7B	36.32	**36.29**	33.31	**31.48**	24.91

than r/h, allowing us to disregard the first term as negligible.

Computational Overhead. Recall that in the standard KV cache algorithm, for each head, we need to do two $1 \times \frac{d}{h}$ vectors and $\frac{d}{h} \times s$ matrix multiplications in the stage of calculating attention and no extra computation in the stage of storing the key and value of the new token. While in our algorithm, since we use low-rank matrices A_K, B_K, A_V, B_V to approximate the K and V matrices respectively, we need two multiplications between $1 \times \frac{d}{h}$ vector and $\frac{d}{h} \times \frac{r}{h}$ matrix, and two multiplications between $1 \times \frac{r}{h}$ vector and $\frac{r}{h} \times s$ matrix in the stage of calculating attention, in the stage of storing the key and value of the new token, we need two multiplications between $1 \times \frac{d}{h}$ vector and $\frac{d}{h} \times \frac{r}{h}$ matrix.

Denote $\Gamma_{computation}$ as the storage ratio of the standard KV cache to our algorithm, we can get

$$\Gamma_{computation} = \frac{h(2\frac{dr}{h^2} + 2\frac{rs}{h} + 2\frac{dr}{h^2})}{2h\frac{d}{h}s} = \frac{2r}{sh} + \frac{r}{d} \approx \frac{r}{d}.$$

J Comparison with low-rank alternatives

AURA-KV belongs to the same broad family as low-rank KV-cache methods, but the comparison in Table 9 highlights why the design details matter. Eigen Attention also performs attention in a reduced subspace, yet it applies a fixed low-rank treatment that does not adapt the rank budget to the layer-wise cache spectra. Under the same 70% budget, AURA-KV improves the average LongBench score over Eigen Attention by $+4.59$ on Llama-3-8B, $+3.40$ on Mistral-7B, and $+2.98$ on LongChat-7B. The gap becomes larger at the 50% budget, where uniform or poorly allocated feature compression has less spare capacity to absorb approximation error.

This comparison also clarifies the role of post-RoPE compression. Methods that compress a pre-RoPE representation must still account for rotary-position transformation during decoding, whereas AURA-KV factorizes the cache after RoPE has already been applied. As a result, the decode path can operate directly on A_K, B_K, A_V, B_V without reconstructing full-dimensional rotated keys and values. Together with the adaptive allocation in Table 5, this gives AURA-KV a practical advantage over feature-dimension compression schemes that use a single rank policy across layers.

K AURA-KV with Other Compression Rate and Benchmarks

Table 10 stress-tests additional rank budgets, keeping only 40% or 60% of the original cache. The 60% budget remains close to the full-cache LongBench averages for Mistral-7B, LongChat-7B, and Llama-3-8B, while the 40% budget shows the expected degradation under aggressive compression. These results are best read as budget-sensitivity evidence: AURA-KV provides a usable quality-memory trade-off, but very small ranks are more task- and model-dependent.

Table 11 provides additional robustness checks on RULER and InfiniteBench:

- At the 70% budget, RULER scores are essentially unchanged across the three models, with only small fluctuations around the full-cache scores.
- InfiniteBench follows the same pattern at 70%, with equal or slightly higher scores in the reported runs.
- At the 50% budget, degradation becomes visible, especially on RULER, but the results remain informative about the operating range of the method rather than undermining the moderate-compression setting.

Together with LongBench, these results support the main claim for moderate compression and clarify the quality trade-off at more aggressive budgets.

L The Performance of Palu

Compared with our AURA-KV + GEAR results in Table 6, the Palu-based compression in Table 12 sacrifices noticeably more accuracy at similar or even milder cache budgets. These numbers show that, for the same memory savings, AURA-KV compression preserves overall performance better than Palu.

Table 10: Baseline results (%) of AURA-KV compression at 40 % and 60 % budgets.

Model	Method	Multi-QA	Single-QA	Summ.	Few-Shot	Code	Synth.	Avg.
Mistral-7B	AURA-KV-40%	30.27	39.15	29.45	67.64	50.13	29.41	37.91
	AURA-KV-60%	35.13	33.93	25.31	69.43	58.42	42.72	44.73
Longchat-7B	AURA-KV-40%	21.93	21.00	13.31	55.13	43.14	6.91	27.15
	AURA-KV-60%	24.00	28.41	24.13	63.24	55.18	13.12	34.11
Llama-3-8B	AURA-KV-40%	22.73	29.91	20.83	61.13	36.11	25.13	34.21
	AURA-KV-60%	33.72	33.21	23.71	66.73	52.31	36.31	40.93

Table 11: RULER and InfiniteBench scores (%) after KV-cache compression.

Model	Method	RULER	InfiniteBench
Mistral-7B	Baseline	68.40	22.81
	AURA-KV-70%	68.30	22.93
	AURA-KV-50%	63.10	19.31
Longchat-7B	Baseline	49.10	15.31
	AURA-KV-70%	49.20	15.42
	AURA-KV-50%	45.80	13.14
Llama-3-8B	Baseline	87.90	23.01
	AURA-KV-70%	88.10	23.06
	AURA-KV-50%	81.20	21.10

M Compatibility with Existing Methods for Mistral

Table 13 shows AURA-KV performance on Mistral-7B-Instruct when it is combined with PyramidKV, SnapKV , H2O, and StreamingLLM, all using the same maximum cache size of 2048 tokens. Using Full KV as the reference (*Avg.* = 45.32), we observe that incorporating our proposed AURA-KV with token eviction schemes usually preserves most of the original performance, with some small positive deltas in the reported runs. Specifically, coupling with **PyramidKV** yields the strongest result: at 70% rank, it reaches *47.57*(+2.25), above the full KV 70%-rank baseline (46.52). When combined with **StreamingLLM**, the 70%-rank model attains an average of *42.70*, only 2.62 points below the reference. For **SnapKV**, the conservative 70%-rank setting records *40.90*, a moderate decrease of 4.42 given its aggressive eviction policy, while **H2O** changes from *39.95* at 100% rank to *40.63* at 70% rank. These results indicate that, in Mistral-7B-Instruct, AURA-KV can be combined with token-eviction strategies to obtain additional cache savings while keeping accuracy in a practical range.

N Combination with KV Cache Quantization Techniques

We combined KV-cache quantization techniques (Zirui Liu et al., 2023; Kang et al., 2024) with our AURA-KV algorithm, and the results on LongBench are shown in Table 14. Table 14 indicates that when we couple AURA-KV compression with the KIVI baseline (AURA-KV + KIVI) and keep only 70 % of the cache, the average LongBench score remains close to the full-cache model; replacing KIVI with GEAR (AURA-KV + GEAR) yields a further small positive delta in this setting. At the harsher 50 % budget, AURA-KV + GEAR retains noticeably more accuracy than AURA-KV + KIVI, suggesting that quantization and feature-dimension compression are compatible when the second stage is chosen carefully.

O LowRankSolver Implementation Details

In Algorithm 1, LowRankSolver is called once after prefill for each attention head and for keys and values separately. The input is a post-RoPE transposed cache matrix $X \in \mathbb{R}^{d_h \times s}$, where d_h is the head dimension and s is the prefill length. The solver returns a basis $A \in \mathbb{R}^{d_h \times r}$ and token-wise coefficients $B \in \mathbb{R}^{r \times s}$ such that $X \approx AB$. We implement this operation with a randomized SVD

Table 12: Performance (%) of SnapKV versus Pyramid KV under two Palu compression ratios.

Model	**Method**	**KV Cache**	Multi-QA	Single-QA	Summ.	Few-Shot	Code	Synth.	Avg.
Mistral-7B	Palu-70%	SnapKV	36.76	37.64	25.13	62.23	55.51	41.80	42.89
		Pyramid KV	37.13	38.18	27.32	68.13	53.14	40.13	43.04
	Palu-50%	SnapKV	30.14	35.81	23.81	58.14	51.13	31.14	39.93
		Pyramid KV	32.73	36.23	24.73	63.43	52.62	32.52	40.23
Longchat-7B	Palu-70%	SnapKV	22.93	28.14	22.41	60.73	58.24	13.24	34.90
		Pyramid KV	22.51	29.31	25.88	61.86	58.13	14.11	35.10
	Palu-50%	SnapKV	22.18	23.19	18.24	58.85	43.91	4.58	27.11
		Pyramid KV	22.53	24.19	21.41	60.11	42.56	5.21	28.51
Llama-3-8B	Palu-70%	SnapKV	24.14	28.96	22.67	63.71	38.61	29.67	33.21
		Pyramid KV	25.01	29.91	23.79	69.79	39.97	31.76	36.16
	Palu-50%	SnapKV	24.19	26.64	20.97	61.83	15.13	5.10	24.15
		Pyramid KV	25.19	28.10	20.74	64.61	14.18	6.40	26.58

Table 13: Performance comparison of our method combined with PyramidKV (PKV), SnapKV (SKV), H2O, StreamingLLM (SLM) (KV Size = 2048), and FullKV (FKV) on LongBench for Mistral-7B. R& Q stands for Low rank compression and Quantization. Our approach, when combined with KV size reduction methods, significantly reduces storage requirements with minimal performance degradation.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
	18409	3619	4559	9151	4887	11214	8734	10614	2113	5177	8209	6258	11141	9289	1235	4206	
Mistral-7B-Instruct, Full KV																	
100%Rank	**30.35**	36.25	**52.00**	46.21	**38.17**	24.01	24.86	23.30	27.36	**75.0**	**89.34**	43.72	**6.00**	94.0	57.73	56.93	45.32
70%Rank	28.41	**39.27**	51.78	**49.41**	38.07	**27.63**	**29.11**	**24.33**	26.52	74.5	88.27	45.20	3.00	**97.0**	**60.84**	**60.97**	**46.52**
50%Rank	27.10	32.45	39.18	44.36	34.77	23.61	21.42	24.18	**27.69**	74.0	88.71	45.52	3.53	64.5	59.75	54.99	41.61
50%R&Q	27.98	32.75	39.50	42.03	34.47	23.52	21.29	24.35	27.05	74.0	88.71	**45.41**	4.44	63.0	59.56	54.89	41.55
Mistral-7B-Instruct, Pyramid KV, KV Size=2048																	
100%Rank	29.41	**40.27**	**52.78**	**50.41**	39.07	28.63	**30.11**	25.33	27.52	75.5	89.27	46.20	4.00	98.0	**61.84**	61.97	47.51
70%Rank	**29.45**	40.12	52.56	50.29	**39.87**	**28.80**	29.99	**25.36**	**27.58**	**75.5**	**89.27**	**46.57**	4.00	**98.0**	61.48	**62.20**	**47.57**
50%Rank	26.90	32.04	49.28	44.86	33.44	23.09	13.23	21.35	19.9	74.0	89.14	40.51	6.41	90.0	53.73	48.35	41.64
50%R&Q	27.14	31.88	48.35	44.73	33.41	23.84	13.21	21.56	19.98	74.0	89.14	40.63	**6.50**	90.0	53.4	48.39	41.63
Mistral-7B-Instruct, SnapKV, KV Size=2048																	
100%Rank	**25.89**	32.93	**48.56**	**42.96**	27.42	19.02	**26.56**	**24.47**	**26.69**	70.0	86.27	**42.57**	5.50	**88.90**	50.42	**46.72**	**41.56**
70%Rank	24.16	**35.43**	38.75	40.66	**34.63**	**20.99**	26.00	22.91	25.99	**72.0**	**88.80**	40.37	5.47	76.15	**58.60**	43.48	40.90
50%Rank	24.41	27.62	38.15	40.47	33.48	18.20	20.42	22.40	21.41	68.0	87.20	40.10	**6.00**	75.00	36.67	33.84	37.08
50%R&Q	24.39	26.63	37.65	40.32	32.87	18.11	20.47	22.12	21.41	68.0	86.70	39.64	6.00	75.15	36.18	32.99	36.79
Mistral-7B-Instruct, H2O, KV Size=2048																	
100%Rank	25.76	**31.10**	49.03	40.76	**26.52**	17.07	24.81	23.64	26.60	**55.0**	86.35	42.48	5.50	88.15	49.93	46.57	39.95
70%Rank	**25.80**	29.01	**49.59**	**42.89**	25.32	**21.53**	**25.67**	**24.22**	**26.61**	53.0	**86.56**	**44.10**	5.30	**89.25**	**50.87**	**46.74**	**40.63**
50%Rank	24.53	28.58	47.65	42.87	25.93	18.42	20.37	13.47	25.74	42.5	85.25	43.26	**6.00**	79.00	42.55	30.38	36.03
50%R&Q	24.33	28.55	47.35	41.85	25.64	17.96	20.21	12.78	25.14	42.5	85.25	42.87	5.58	79.83	41.97	29.12	35.68
Mistral-7B-Instruct, StreamingLLM, KV Size=2048																	
100%Rank	25.68	**34.05**	37.73	44.59	36.17	21.85	25.92	23.29	27.01	**73.0**	**89.95**	41.74	**4.90**	69.33	61.27	54.63	41.94
70%Rank	**25.91**	30.03	**37.75**	**44.60**	**36.50**	**22.18**	**27.60**	**23.69**	**27.04**	70.5	89.07	**45.93**	4.50	**79.00**	**62.20**	**56.76**	**42.70**
50%Rank	22.12	26.07	36.35	42.96	29.40	18.14	10.30	20.13	19.81	70.5	88.89	40.06	4.89	65.25	54.43	48.39	37.35
50%R&Q	21.89	25.93	35.25	42.45	29.38	18.04	9.75	20.10	19.63	70.5	88.75	39.74	4.43	64.27	53.42	48.21	36.98

Table 14: Performance (%) of KV compression baselines and our algorithms on six task groups, and higher is better.

Model	Method	Multi-QA	Single-QA	Summ.	Few-Shot	Code	Synth.	Avg.
Mistral-7B	AURA-KV-70% +KIVI	36.34	39.15	25.13	69.34	59.03	46.20	45.97
	+GEAR	38.36	39.51	26.66	69.30	60.87	48.00	46.12
	AURA-KV-50% +KIVI	32.30	33.00	24.41	68.91	56.13	32.32	41.31
	+GEAR	34.21	32.93	24.23	69.20	57.12	33.92	41.57
Longchat-7B	AURA-KV-70% +KIVI	23.19	27.00	26.32	63.30	56.98	14.70	35.35
	+GEAR	24.27	30.90	26.63	63.32	77.10	14.90	36.52
	AURA-KV-50% +KIVI	22.70	25.13	15.90	61.42	48.13	9.13	30.91
	+GEAR	23.31	26.32	16.30	60.97	49.18	10.16	31.32
Llama-3-8B	AURA-KV-70% +KIVI	32.13	31.21	23.13	66.40	56.11	37.12	41.01
	+GEAR	35.13	33.37	24.51	67.00	56.73	37.14	41.32
	AURA-KV-50% +KIVI	30.12	30.14	22.74	65.71	45.31	38.13	38.13
	+GEAR	31.72	33.21	22.43	65.50	45.40	33.28	38.54

Algorithm 2 LowRankSolver for post-RoPE cache matrices

1: **Input:** post-RoPE cache matrix $X \in \mathbb{R}^{d_h \times s}$, target rank r
2: **Output:** basis $A \in \mathbb{R}^{d_h \times r}$, coefficients $B \in \mathbb{R}^{r \times s}$
3: Draw a Gaussian random matrix $\Omega \in \mathbb{R}^{s \times r}$.
4: Form the range sketch $Y = X\Omega \in \mathbb{R}^{d_h \times r}$.
5: Compute a thin QR decomposition $Y = QR$.
6: Set $A = Q[:, 1:r]$.
7: Project the cache into the basis: $B = A^\top X$.
8: **return** A, B

range finder rather than a full deterministic SVD, because the cost is linear in s and the operation is only needed once after prefill.

Implementation notes. For keys, Algorithm 2 is applied to K_i^T and returns A_{K_i}, B_{K_i}; for values, it is applied to V_i^T and returns A_{V_i}, B_{V_i}. The basis matrices are kept fixed during decoding, while each newly generated key/value vector is appended only through its low-dimensional coefficients $A_{K_i}^\top k_{i,t}$ and $A_{V_i}^\top v_{i,t}$. Thus the full-dimensional prefill cache can be released after the solver finishes.

Complexity. The random projection $X\Omega$ and coefficient projection $A^\top X$ each cost $O(d_h s r)$. The thin QR step costs $O(d_h r^2)$, which is smaller in our setting because $r \ll s$. Therefore the one-time prefill compression cost per head and per cache type is $O(d_h s r)$, linear in sequence length. The empirical effect of this operation is reported in Table 7.

P Clarification on Adaptive Rank Selection

To make adaptive rank selection reproducible, we give the exact implementation used to obtain the key/value ranks in Table 6. For each model, we first collect post-RoPE prefill caches on an unlabeled calibration split and compute singular-value spectra for every layer and cache type $c \in \{K, V\}$. In the reported experiments, one key rank $r_{\ell,K}$ and one value rank $r_{\ell,V}$ are assigned to all heads in the same layer ℓ; the per-head routine in Algorithm 1 therefore receives these layer-level ranks for every head in that layer.

Rank from retained energy. For a candidate information-retention threshold τ, the selected rank is the smallest rank whose cumulative spectral energy reaches τ:

$$r_{\ell,c}(\tau) = \min \left\{ r : \frac{\sum_{i=1}^{r} \sigma_{\ell,c,i}^2}{\sum_{j=1}^{d_h} \sigma_{\ell,c,j}^2} \geq \tau \right\}, \quad (38)$$

where $\sigma_{\ell,c,i}$ is the i-th singular value of the calibration cache matrix for layer ℓ and cache type c.

Budget check. The search uses the retained-rank ratio

$$\widehat{\rho}(\tau) = \frac{\sum_{\ell=1}^{L} (r_{\ell,K}(\tau) + r_{\ell,V}(\tau))}{2Ld_h}. \quad (39)$$

A candidate is feasible if $\widehat{\rho}(\tau) \leq p_{\text{Comp}}$. For finite sequence length s, the exact stored-cache ratio after factorization is

$$\rho_{\text{mem}}(\tau; s) = \frac{\sum_{\ell=1}^{L} \sum_{c \in \{K,V\}} h(s + d_h) r_{\ell,c}(\tau)}{2Lhsd_h}, \quad (40)$$

Algorithm 3 ADAPTIVERANKSEARCH

Input: spectra $\{\sigma_{\ell,c}\}_{\ell=1,c\in\{K,V\}}^{L}$, rank budget p_{Comp}, grid $\mathcal{T} = \{0.99, 0.98, \ldots\}$
Output: layer-wise ranks $\{r_{\ell,K}, r_{\ell,V}\}_{\ell=1}^{L}$
for each τ in $\mathcal{T}$ from high to low **do**
  for each layer ℓ and cache type $c \in \{K, V\}$ **do**
    Compute $r_{\ell,c}(\tau)$ by Eq. 38.
  end for
  Compute $\widehat{\rho}(\tau)$.
  if $\widehat{\rho}(\tau) \leq p_{\text{Comp}}$ **then**
    Set $r_{\ell,c} = r_{\ell,c}(\tau)$ for all ℓ, c.
    Stop the search.
  end if
end for
return $\{r_{\ell,K}, r_{\ell,V}\}_{\ell=1}^{L}$

where the numerator accounts for both the basis $A \in \mathbb{R}^{d_h \times r}$ and the coefficient matrix $B \in \mathbb{R}^{r \times s}$. Because long-context prefill has $s \gg d_h$, $\widehat{\rho}$ closely tracks the dominant coefficient-cache cost, while ρ_{mem} is used when reporting actual memory and prefill overhead.

The first feasible threshold is the largest feasible τ because the grid is searched from high to low. This gives the least aggressive truncation that still satisfies the global budget. In all reported experiments, the grid step is 0.01 and we start from $\tau = 0.99$; the target budgets are the reported retained-rank ratios such as $p_{\text{Comp}} = 0.70$ and 0.50. Ranks are clipped to the valid range $1 \leq r_{\ell,c} \leq d_h$. The calibration pass uses unlabeled prefill caches only and does not update model weights. Table 6 reports representative ranks, and Fig. 6 plots the full layer-wise schedules used in the experiments.

Q Additional Experiments and Baseline Comparisons

We conducted additional experiments on newer model versions and added controlled comparisons with closely related low-rank KV-cache methods.

Q.1 Evaluation on Newer Model Versions

We evaluated `Llama-3.1-8B-Instruct` and `Mistral-7B-Instruct-v0.3` on the LongBench benchmark (Bai et al., 2024). Results are reported in Table 15. These runs extend the main Llama-3 and Mistral experiments to more recent checkpoints. We have not yet run Qwen-family models, and therefore do not claim coverage of that model family.

Observations.

- **Robustness.** AURA-KV remains robust on these newer architectures.
- **Small positive deltas.** On `Llama-3.1-8B-Instruct`, AURA-KV-70% slightly improves the average score (+0.20) over the full baseline. We interpret this as robustness under compression rather than conclusive evidence of a systematic accuracy improvement.
- **Low degradation.** On `Mistral-7B-Instruct-v0.3`, the performance drop at 70% budget is minor (≈0.94 points).

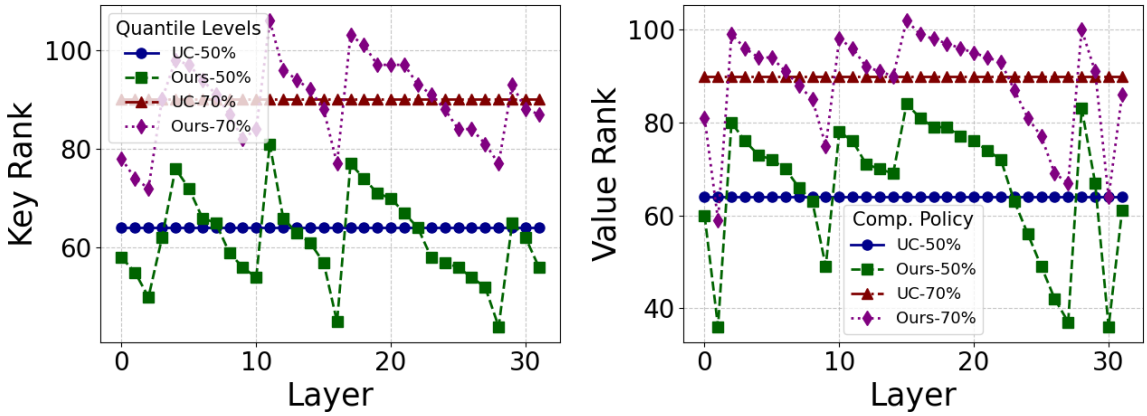
Q.2 Scope of Related Low-Rank Baselines

The main paper reports controlled numerical comparisons with Palu and Eigen Attention under the same rank budgets and benchmarks. ThinK (Xu et al., 2024) and LoRC (Zhang et al., 2024) are also closely related because they reduce the feature dimension of KV representations. However, their compression mechanisms are not identical to a fixed post-prefill subspace factorization: ThinK performs query-driven channel pruning, while LoRC uses a progressive compression strategy. We therefore avoid mixing unmatched numbers from different kernels, hardware, and cache-update policies. Instead, we compare against the low-rank baselines that can be run under the same retained-rank setting, and discuss ThinK and LoRC qualitatively in Section J. Adding a same-hardware reproduction of ThinK and LoRC is an important next step for a broader empirical comparison.

R Latency and Computational Cost of Prefill Compression

R.1 Theoretical Computational Cost

As clarified in Appendix O, our LOWRANKSOLVER is implemented using randomized SVD, whose dominant cost is $O(d_h s r)$ per head and per cache type, where d_h is the head dimension, s is the sequence length, and r is the target rank. Crucially, this cost is *linear* in s, ensuring scalability to long-context prefill.



(a) Key ranks across layers.

(b) Value ranks across layers.

Figure 6: Full layer-wise rank schedules selected by AURA-KV. UC denotes the uniform-compression baseline, while Ours denotes the adaptive policy used by AURA-KV.

Table 15: LongBench results on newer model versions. Higher is better.

Model	Method	Multi-QA	Single-QA	Summ.	Few-Shot	Code	Synth.	Avg.
Mistral-7B-Instruct-v0.3	Baseline	41.30	38.95	27.17	70.70	51.75	49.13	46.01
	AURA-KV-70%	42.06	37.11	25.96	68.30	50.83	49.57	45.07
	AURA-KV-50%	39.31	35.70	23.27	60.27	45.29	43.92	40.88
Llama-3.1-8B-Instruct	Baseline	25.28	23.70	28.37	69.43	53.02	56.62	41.27
	AURA-KV-70%	25.34	23.83	28.51	69.10	54.43	56.97	41.47
	AURA-KV-50%	22.73	19.34	25.17	60.37	47.43	51.69	36.12

Table 16: Coverage of related KV-cache compression baselines.

Family	Methods covered	How used in this paper
Low-rank feature compression	Palu, Eigen Attention	Direct numerical comparison under matched rank budgets.
Token eviction	H2O, StreamingLLM, SnapKV, PyramidKV	Compositional experiments with AURA-KV as a second-stage compressor.
KV quantization	KIVI, GEAR	Compositional experiments with feature-rank compression.
Recent feature pruning / progressive compression	ThinK, LoRC	Qualitative comparison; same-hardware reproduction left for future work.

R.2 Empirical Prefill Latency

Table 7 in the main paper compares the standard prefill throughput with the throughput when applying our KV dimension reduction (LowRankSolver) during prefill.

Analysis. The computational overhead introduced by the dimension reduction operation is small, resulting in a ∼4–5% decrease in prefill throughput. Importantly, this overhead is a *one-time* cost incurred only during the prefill phase. In long-generation settings, this initial delay is typically outweighed by the decoding-time latency reduction and cache-memory savings.

S Scope of Head-Dimension Compression

AURA-KV compresses the feature dimension inside each attention head. This means its standalone memory reduction is bounded by the original head dimension d_h; for Llama-3-8B and Mistral-7B, where $d_h = 128$, a 50% retained-rank budget corresponds to roughly a $2\times$ feature-dimension reduction before accounting for the small basis-storage overhead. Token-pruning methods can obtain larger reductions by shortening the sequence dimension, which is why we position AURA-KV as an orthogonal second-stage compressor rather than a replacement for token eviction. The method should become more attractive for architectures with larger KV head dimensions, wider grouped-query heads, or deployments that combine moderate feature compression with token pruning and quantization.

T Fixed-Basis Decode Assumption

AURA-KV estimates A_K and A_V from the prefill cache and keeps these bases fixed during decoding. This design is what makes incremental decoding cheap: every newly generated key/value vector is stored as the least-squares coefficient in

the existing basis, which reduces to $A_{K_i}^\top k_{i,t}$ and $A_{V_i}^\top v_{i,t}$ because the bases are orthonormal. The assumption is therefore not that future tokens are identical to prefill tokens, but that the dominant key/value subspaces remain sufficiently stable for the evaluated generation horizon. All reported results use this fixed-basis path, providing end-to-end evidence that it is adequate for our LongBench, RULER, and InfiniteBench settings. Nevertheless, this is not a complete ablation of basis drift. For very long generations or distribution shifts between prompt and generated content, periodically recomputing or expanding the bases may further reduce approximation error at additional cost. We leave refresh-frequency ablations and error-growth diagnostics to future work.

U Further Discussion about Small Positive Deltas

We do not treat small positive deltas after compression (e.g., +0.34% for Llama-3-8B) as a main claim of the paper. A plausible explanation is redundancy removal: the tail of the singular-value spectrum may contain components that are less useful for the downstream task, so truncation can sometimes behave like a mild denoising operation. This interpretation is also consistent with token-eviction results in Table 4, where PyramidKV shows a very small positive delta over the full-cache setting (41.49 vs. 41.46). However, these effects are small and may depend on benchmark variance, decoding settings, and model-specific spectra. The central empirical claim is therefore preservation of quality under moderate compression, not reliable accuracy improvement from compression. Future work should test this hypothesis with repeated seeds, bootstrap confidence intervals, and controlled truncation studies that isolate spectral denoising from ordinary benchmark variance.

V Reproducibility and Computational Requirements

AURA-KV is training-free and requires no fine-tuning or gradient computation. The only additional computation is the post-prefill LowRank-Solver call for each layer, head, and cache type, followed by coefficient projection for newly generated tokens. The adaptive-rank search uses the grid and memory accounting described in Appendix P; the selected ranks are fixed before evaluation. All latency numbers in Table 1 and Table 7 are measured on the reported A40 setting, and the compressed prefill throughput includes the one-time low-rank factorization cost. We will release the implementation, rank-selection script, and evaluation commands with the paper so that the reported budgets and layer-wise ranks can be reproduced.

W Statement on the Use of AI-Assisted Tools

We used AI-assisted tools to support the writing and editing of this manuscript. Specifically, these tools were employed to help refine grammar, improve readability, and enhance the clarity and consistency of the presentation. All technical content—including the problem formulation, algorithm design, theoretical analysis, experimental setup, results, and conclusions—was developed and verified by the authors. The authors reviewed, validated, and edited all AI-assisted outputs to ensure correctness, originality, and compliance with the venue's policies.