ShallowKV: Plug-and-Play Shallow-Layer KV Cache Compression via Low-Rank Reconstruction

Anonymous ACL submission

Abstract

Long-context large language model inference is increasingly limited by the memory footprint of the key-value (KV) cache. Many existing KV cache compressors apply a retain-or-evict token selection primitive throughout the transformer depth, despite a clear empirical asymmetry: attention is widely scattered in lower layers but increasingly sparse and concentrated on fewer critical tokens in deeper layers. We propose **ShallowKV**, a plug-and-play wrapper for existing compressors that targets these shallow layers by leveraging the strong low-rank structure of their KV states. Specifically, ShallowKV retains salient KV entries as exact anchors and represents the remaining cache via adaptive low-rank factorization, while deeper layers continue to use the original retain-or-evict policy. Experiments on LongBench with Mistral-7B, Llama-2-7B, and Llama-3.1-8B demonstrate that ShallowKV generally enhances strong retention-based baselines under matched KV budgets, achieving better quality-memory trade-offs across multiple model families.

1 Introduction

Large language models (LLMs) based on the Transformer architecture (Vaswani et al., 2017) have greatly improved the ability to process long contexts. This capability enables a wide range of applications, including multi-turn dialogue (Chiang et al., 2023), document summarization (Zhang et al., 2024), question answering (Bai et al., 2024), information retrieval (Zhu et al., 2023), and repository-level code generation (Li et al., 2025).

However, serving such long contexts remains computationally expensive. KV caching reduces repeated computation during autoregressive decoding by storing key-value (KV) states for all processed tokens, but this benefit comes with a growing memory cost. The resulting cache scales linearly with sequence length and can dominate inference memory in long-document, multi-turn, and repository-level workloads, even when hardware-aware attention kernels and serving systems improve computation and memory management (Dao et al., 2022; Pope et al., 2023; Kwon et al., 2023; Zheng et al., 2024; NVIDIA, 2023).

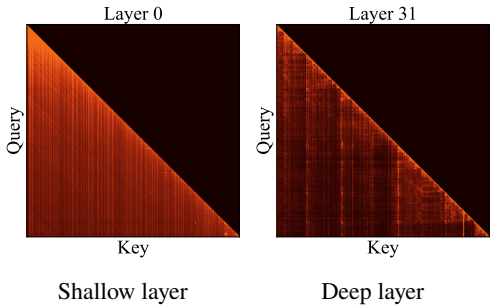
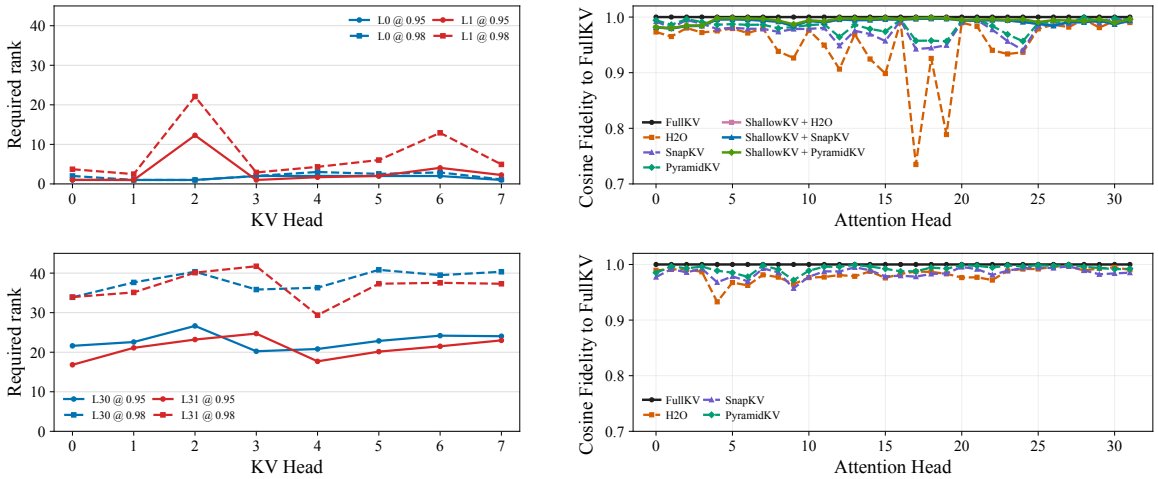


Figure 1: Attention distribution comparison between shallow and deep layers of Mistral-7B on GovReport.

To mitigate this bottleneck, existing inference-time compression methods mainly follow two directions: retention-based methods keep a subset of KV states and evict the rest, as in H2O, SnapKV, and PyramidKV (Zhang et al., 2023; Li et al., 2024; Cai et al., 2024), while representation-compression methods reduce the storage cost of cached states through low-rank or numeric approximations, as in ShadowKV and KV quantization methods (Sun et al., 2024; Hooper et al., 2024; Liu et al., 2024).

Retention-based compressors are attractive for deployment because they are training-free, plug-and-play, and compatible with existing long-context serving pipelines. However, these methods predominantly rely on a binary keep-or-drop primitive: selected tokens are preserved at full precision, while the unselected rest are permanently discarded. While effective in deeper layers—where attention is sparsely concentrated on a few critical tokens—this hard-eviction strategy risks destroy-



(a) Head-wise energy-preserving rank. **(b) Head-wise fidelity to FullKV.**

Figure 2: Layer-wise analysis on GovReport using Mistral-7B. In both panels, the top subfigure shows shallow layers and the bottom subfigure shows deep layers. Panel (a) reports the KV-head-wise rank required to preserve 95%/98% spectral energy (@0.95/@0.98); in this panel, $L0$ denotes layer 0 and analogously for other layer labels. Panel (b) reports head-wise cosine fidelity to FullKV, with shallow-layer values averaged over layers 0–3 and deep-layer values averaged over layers 28–31.

ing widely distributed contextual evidence in shallow layers. As shown in Figure 1, attention in shallow layers is markedly more scattered. Figure 2(b) further reveals that retention-only compression causes severe fidelity fluctuations from FullKV in shallow layers. These observations suggest that strict exact-token retention is sub-optimal for shallow-layer representations.

Recent work such as ShadowKV shows that key states contain exploitable low-rank structure and can be represented with SVD-based projections (Sun et al., 2024). We find that this structure is especially pronounced in shallow layers. As shown in Figure 2(a), shallow-layer KV states require much smaller ranks to preserve the same spectral energy than deeper-layer KV states. Furthermore, as shown in Figure 2(b), the noticeable head-wise fidelity fluctuations caused by retention-only compression relative to FullKV in shallow layers can be mitigated by ShallowKV-enhanced variants. These results suggest that KV cache compression should not only decide *how much* memory to assign to each layer, but also *which compression primitive* each layer should use.

Motivated by this insight, we propose ShallowKV, a plug-and-play module that upgrades existing retention compressors by targeting shallow layers with widely scattered attention. Instead of hard eviction, ShallowKV adopts a hybrid compression primitive. During prefill, it partitions the prompt into semantic chunks, preserving highly relevant local spans as exact anchors while compressing the widely distributed residual keys and values into adaptive-rank SVD factors. During decoding, new states are maintained in a dense local window, and older states are periodically compressed into low-rank chunks using the current generation context. This dual-branch design elegantly balances local precision with global context: the original retention backend and exact anchors maintain salient details, while the low-rank residuals approximately preserve distributed contextual information in shallow layers. A visual comparison of the resulting layer-wise cache storage patterns is illustrated in Figure 3.

Experiments on LongBench with Mistral-7B, Llama-2-7B, and Llama-3.1-8B show that ShallowKV generally improves retention backends under matched KV budgets, winning 71.4% of non-tied task-level comparisons across the main evaluated ratios, models, backends, and tasks. This indicates a better quality-memory trade-off from using low-rank reconstruction in shallow layers.

Our contributions are summarized as follows:

- We demonstrate that shallow layers with widely scattered attention are prone to losing distributed contextual evidence under retain-or-evict KV compression.
- Leveraging the strong low-rank structure of shallow-layer KV states, we propose ShallowKV, a plug-and-play wrapper that pre-

serves salient anchors exactly while reconstructing non-anchor KV states with adaptive-rank SVD.

- We instantiate ShallowKV on three retention backends, H2O, SnapKV, and PyramidKV, and show improved quality-memory trade-offs across LongBench and multiple model families under matched KV budgets.

2 Related Work

KV Cache Compression. Recent work reduces long-context KV-cache memory mainly through token retention or representation compression.

Retention-based methods keep a subset of cached states according to attention scores, token persistence, recent-window signals, or layer-wise cache allocation (Liu et al., 2023; Oren et al., 2024; Adnan et al., 2024; Zhang et al., 2023; Xiao et al., 2023; Ge et al., 2024; Li et al., 2024; Cai et al., 2024; Yang et al., 2024). Recent methods further refine cache allocation with head-level routing, retrieval heads, evolutionary layer budgets, phase-aware compression, semantic chunks, or tree-structured retention (Fu et al., 2024; Tang et al., 2024; Xiao et al., 2025; Yu and Chai, 2025; Wu et al., 2025; Liu et al., 2025; He et al., 2025). These methods are deployment-friendly, but unselected states are ultimately removed. ShallowKV instead preserves shallow-layer residual information with low-rank reconstruction.

Representation-based methods reduce the storage cost of cached states themselves. KVQuant, KIVI, QServe, and recent quantization methods compress KV tensors with low-bit, transformed, or hardware-aware representations (Hooper et al., 2024; Liu et al., 2024; Lin et al., 2025; Wang et al., 2025; Saxena and Roy, 2025; Zandieh et al., 2025; Kim et al., 2025). Other work explores architectural latent caches, low-rank keys, or inter-layer attention similarity (DeepSeek-AI, 2024; Singhania et al., 2024; Ma et al., 2024). ShadowKV also uses SVD, but mainly as a retrieval index, whereas ShallowKV stores SVD factors as the compressed cache in shallow layers (Sun et al., 2024).

3 Method

3.1 Problem Setup

Consider an autoregressive transformer with L layers. For layer l, let $\mathbf{K}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{H_k \times T \times d}$ denote

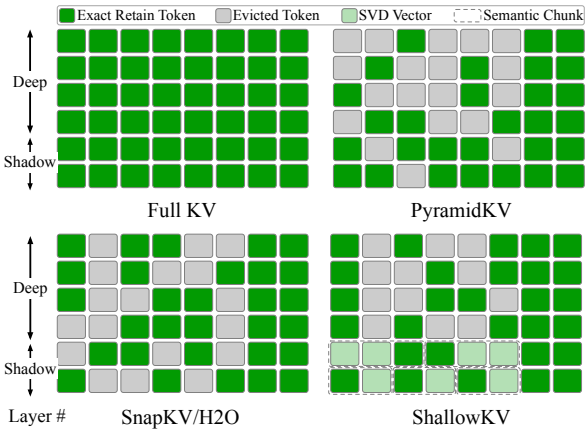


Figure 3: Layer-wise KV cache storage patterns.

the key and value cache for a prompt of length T, where H_k is the number of KV heads and d is the head dimension. The full cache memory is proportional to $2LH_kTd$.

Our goal is to reduce the cache memory while preserving downstream generation quality.

Rather than replacing the KV compression policy in all transformer layers, ShallowKV applies its low-rank module only to a configurable set of shallow layers and leaves deeper layers to an existing retention-based compressor.

Formally, let $\mathcal{L}_\mathrm{s}$ denote the shallow layers routed to ShallowKV, and let $\mathcal{L}_\mathrm{d}$ denote the deeper layers routed to an existing retention compressor. For layer l, the compressed cache is

$$\mathcal{M}^{(l)} = \begin{cases} C_\mathrm{s}(\mathbf{K}^{(l)}, \mathbf{V}^{(l)}), & l \in \mathcal{L}_\mathrm{s}, \\ C_\mathrm{r}(\mathbf{K}^{(l)}, \mathbf{V}^{(l)}), & l \in \mathcal{L}_\mathrm{d}. \end{cases} \tag{1}$$

Figure 4 provides an overview of ShallowKV. **The complete procedural pipeline, comprising semantic chunking, prefill-stage compression, and decode-side recompression, is detailed in Algorithms 1, 2, 3, and 4 in Appendix C.**

3.2 Prefill-Stage Compression

During prefill, ShallowKV builds the initial compressed cache for shallow layers. It partitions the prompt into semantic-boundary chunks, augments recent-query scoring with representative prompt queries, and applies exact-anchor preservation plus low-rank residual compression within each chunk.

Semantic chunk prior. ShallowKV partitions the middle prompt region into contiguous chunks $\mathcal{C} = \{c_1, \ldots, c_N\}$ using a lightweight semantic-boundary prior. The prior keeps chunk lengths within a bounded range, with a target size of 160

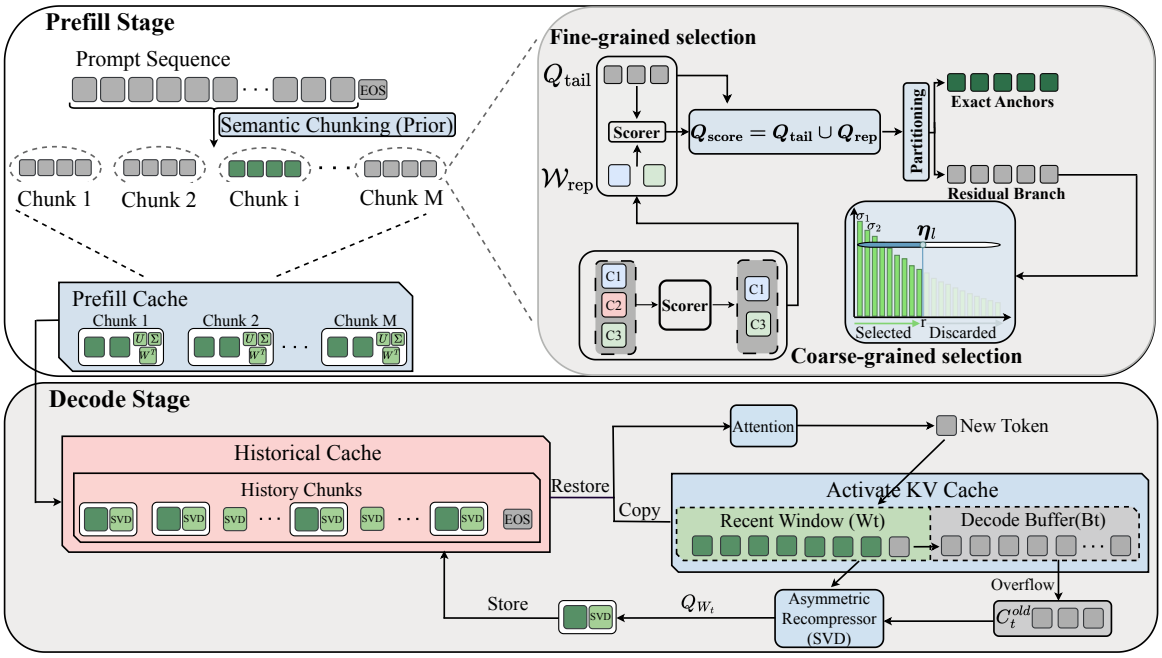


Figure 4: Overview of ShallowKV. ShallowKV wraps the shallow layers of an existing retention compressor. During prefill, it organizes the prompt into semantic-boundary chunks, selects exact anchors with representative-enhanced scoring, and compresses non-anchor KV states into adaptive-rank SVD factors. During decoding, new token states enter the recent window, and evicted old states are appended to the decode buffer. Whenever a chunk in the decode buffer becomes full, ShallowKV compresses that chunk using the current local window as the scoring context.

tokens and a valid range of 96–256 tokens, so that boundaries tend to align with natural textual or structural transitions. We provide the implementation procedure in Appendix C.1.

Representative-enhanced prefill scoring. For each shallow layer, ShallowKV scores tokens within each chunk using both recent and representative prompt queries. Let $\mathbf{Q}_{\text{tail}}$ denote the queries from the recent prompt suffix. To avoid relying only on recency, we construct a representative query set $\mathbf{Q}_{\text{rep}}$ from earlier prompt regions.

Using the semantic chunks defined above, we first score each chunk by its relevance to $\mathbf{Q}_{\text{tail}}$ with cosine similarity between normalized query states. We keep the high-scoring chunks as the coarse window set $\mathcal{W}_{\text{rep}}$. Let $\mathbf{K}_{\mathcal{W}_{\text{rep}}}$ denote the key states in these selected chunks. ShallowKV then scores them with $\mathbf{Q}_{\text{tail}}\mathbf{K}_{\mathcal{W}_{\text{rep}}}^{\top}$ and selects the corresponding query states at the high-scoring positions to form $\mathbf{Q}_{\text{rep}}$. The scoring queries for chunk c are therefore

$$\mathbf{Q}_{\text{score}}(c) = [\mathbf{Q}_{\text{tail}}; \mathbf{Q}_{\text{rep}}(c)]. \tag{2}$$

Given $\mathbf{Q}_{\text{score}}(c)$ and chunk keys $\mathbf{K}_c$, ShallowKV computes attention scores

$$\mathbf{A}_c = \text{softmax}\left(\frac{\mathbf{Q}_{\text{score}}(c)\mathbf{K}_c^{\top}}{\sqrt{d}}\right). \tag{3}$$

We reduce $\mathbf{A}_c$ over query positions and heads to obtain a scalar importance score a_i for each token i in the chunk. This representative-enhanced scoring gives each chunk access to prompt-wide evidence while still preserving the local relevance signal from recent queries.

Exact anchors and low-rank residuals. After obtaining token scores $\{a_i\}_{i\in c}$, ShallowKV partitions each chunk into exact anchors and residual states for low-rank compression. The exact branch stores high-scoring local spans rather than isolated tokens. Concretely, we select subspans according to the token scores and preserve the corresponding keys and values exactly:

$$\mathcal{E}_c = \text{ExactSelect}(\{a_i\}_{i\in c}), \quad \mathcal{R}_c = c \setminus \mathcal{E}_c. \tag{4}$$

This keeps anchors locally coherent and avoids exact-cache fragmentation.

The remaining residual positions $\mathcal{R}_c$ are compressed by SVD. For the residual key and value matrices, ShallowKV stores truncated factors:

$$\mathbf{K}_{\mathcal{R}_c} \approx \mathbf{U}_K^{(c)}\mathbf{B}_K^{(c)}, \tag{5}$$

$$\mathbf{V}_{\mathcal{R}_c} \approx \mathbf{U}_V^{(c)}\mathbf{B}_V^{(c)}, \tag{6}$$

where the singular values are folded into the right factor $\mathbf{B} = \mathbf{\Sigma}\mathbf{W}^\top$. Thus, residual states are stored as $\mathbf{U}$ and $\mathbf{B}$ factors instead of dense tokens.

ShallowKV uses adaptive rank selection for the residual branch. For each chunk and layer, it chooses the smallest rank r_c whose retained spectral energy exceeds the layer-specific threshold:

$$r_c = \min\left\{ r : \frac{\sum_{j=1}^{r} \sigma_j^2}{\sum_j \sigma_j^2} \geq \eta_l \right\}. \tag{7}$$

3.3 Decode-Side Compression

During decoding, ShallowKV attends over SVD-compressed historical chunks together with the dense recent window. Our implementation evaluates historical chunks directly from their SVD factors with a chunk-wise CUDA kernel, avoiding full KV materialization at each decoding step. After attention produces a new token, the newly generated KV states are stored densely in a recent window $\mathbf{W}_t$, whose token index set is denoted by $\mathcal{W}_t$, so that the latest tokens remain available at full fidelity. When $\mathbf{W}_t$ exceeds its capacity, the oldest generated state is moved into a decode buffer $\mathcal{B}_t$:

$$\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{\mathbf{k}_{\text{old}}, \mathbf{v}_{\text{old}}\}. \tag{8}$$

Once $\mathcal{B}_t$ reaches capacity, ShallowKV compresses the oldest buffered chunk $\mathcal{C}_t^{\text{old}} \subseteq \mathcal{B}_t$.

Decode-side scoring differs from prefill scoring. In prefill, representative prompt queries are used to improve prompt-wide coverage. During decoding, however, the relevant scoring signal is local to the current generation window. Therefore, ShallowKV uses the recent-window queries $\mathbf{Q}_{\mathcal{W}_t}$ to score the buffered states:

$$\mathbf{A}_{\text{dec}}^{(t)} = \text{softmax}\left(\frac{\mathbf{Q}_{\mathcal{W}_t} \mathbf{K}_{\mathcal{C}_t^{\text{old}}}^\top}{\sqrt{d}} \right). \tag{9}$$

The same exact-anchor plus adaptive low-rank compression rule is applied to $\mathcal{C}_t^{\text{old}}$:

$$\hat{\mathbf{K}}_{\mathcal{C}_t^{\text{old}}}, \hat{\mathbf{V}}_{\mathcal{C}_t^{\text{old}}} = C_{\text{svd}}\left(\mathbf{K}_{\mathcal{C}_t^{\text{old}}}, \mathbf{V}_{\mathcal{C}_t^{\text{old}}}; \mathbf{Q}_{\mathcal{W}_t} \right). \tag{10}$$

After compression, the compressed chunk is registered into the decode-side historical cache and removed from $\mathcal{B}_t$.

RoPE-aware score reuse. Consider two 2D SVD bases $\mathbf{b}_1, \mathbf{b}_2$ and a compressed key $\mathbf{k}_i = \lambda_{i,1}\mathbf{b}_1 + \lambda_{i,2}\mathbf{b}_2$. For token position p_i, let $\theta_i = p_i\omega$. For one basis $\mathbf{b}_j = (b_{j,1}, b_{j,2})$,

$$\begin{aligned} \mathbf{q}^\top \mathbf{R}(\theta_i)\mathbf{b}_j = &(q_1 b_{j,1} + q_2 b_{j,2}) \cos\theta_i \\ &+ (q_2 b_{j,1} - q_1 b_{j,2}) \sin\theta_i. \end{aligned} \tag{11}$$

Define the reusable query-basis terms

$$A_j = q_1 b_{j,1} + q_2 b_{j,2},\ B_j = q_2 b_{j,1} - q_1 b_{j,2}. \tag{12}$$

Then the score can be computed without reconstructing $\mathbf{k}_i$:

$$s_i = \tau \sum_{j=1}^{2} \lambda_{i,j} \left(A_j \cos\theta_i + B_j \sin\theta_i \right). \tag{13}$$

Thus, (A_j, B_j) are reused across tokens, while each token only injects its own RoPE position through $\cos\theta_i$ and $\sin\theta_i$. More details and equivalence analysis are provided in Appendices C.5.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate ShallowKV on LongBench across QA, summarization, few-shot learning, synthetic retrieval, and code completion.

Models. We use Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) as the main model and additionally evaluate Llama-2-7b-chat-hf (Touvron et al., 2023) and Llama-3.1-8B-Instruct (Meta AI, 2024). For brevity, we refer to them as Mistral-7B, Llama-2-7B, and Llama-3.1-8B in the rest of the paper.

Baselines. We compare against FullKV and three representative retention-based KV cache compressors: H2O, SnapKV, and PyramidKV. FullKV stores all key and value states and serves as the reference setting. For ShallowKV, we instantiate three hybrid variants, denoted ShallowKV + H2O, ShallowKV + SnapKV, and ShallowKV + PyramidKV, where shallow layers are routed to ShallowKV and deeper layers use the corresponding retention compressor.

Compression ratios. We evaluate compressed methods under aggressive KV budgets. For LongBench, we report retention ratios of 0.4/0.2 on Mistral-7B and Llama-2-7B, plus 0.3 for Mistral ablations; for Llama-3.1-8B, we report 0.4. To ensure matched total KV memory, we slightly reduce the backend retention ratio to offset the overhead of exact anchors and SVD factors(e.g., decreasing the ratio from 0.4 to 0.38, from 0.3 to 0.27, and from 0.2 to 0.16). Thus, all methods are compared under identical overall memory constraints.

Method	Ratio	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		Avg.
		NQA	Qasper	MFQA	Hotpot	2Wiki	Musique	Gov.	QMSum	MNews	TREC	Trivia	SAMSum	PCount	PR	LCC	Repo	
FullKV	1.0	27.20	32.56	49.28	43.64	27.53	19.21	31.87	24.48	27.10	71.00	86.33	43.36	3.15	86.56	57.41	56.20	42.93
H2O	0.4	25.35	29.77	43.64	43.16	25.86	17.59	32.73	24.05	28.51	71.00	85.33	40.41	3.07	86.54	54.12	47.30	41.15
SnapKV		27.00	31.15	49.47	42.98	27.97	18.93	31.01	**24.59**	25.63	71.00	86.06	43.29	2.42	85.89	57.37	50.23	42.19
PyramidKV		26.66	**31.85**	49.18	41.20	27.88	18.52	30.54	24.27	26.17	71.00	86.06	43.02	2.59	85.93	56.58	49.99	41.97
ShallowKV + H2O		26.24	29.37	44.92	**44.01**	26.75	18.55	**33.02**	23.81	**28.59**	**71.50**	**86.50**	41.05	**3.35**	86.39	53.74	46.71	41.53
ShallowKV + SnapKV		**27.12**	31.47	**49.73**	**44.01**	**28.15**	**18.99**	31.68	24.28	25.79	71.00	86.43	**43.41**	3.13	85.39	**57.44**	50.20	**42.39**
ShallowKV + PyramidKV		26.84	31.82	49.49	43.70	27.91	18.36	30.71	24.55	26.21	71.00	86.21	42.96	2.86	**86.77**	56.37	**50.68**	42.28
H2O	0.2	24.48	26.46	39.75	41.02	23.27	17.24	31.89	23.11	28.47	**71.50**	85.91	39.23	2.86	80.23	50.17	45.06	39.42
SnapKV		**26.42**	30.70	49.19	41.97	**27.97**	18.72	29.48	24.14	24.03	70.00	86.24	**43.55**	2.39	**86.06**	55.63	49.35	41.62
PyramidKV		25.88	29.71	49.51	40.91	27.02	18.75	28.71	24.02	24.39	70.50	**86.79**	42.77	2.21	84.56	55.81	49.14	41.29
ShallowKV + H2O		24.60	27.14	40.93	41.74	23.94	17.39	**32.03**	22.84	**28.58**	**71.50**	86.02	38.39	3.24	81.08	50.10	45.03	39.66
ShallowKV + SnapKV		26.22	**31.12**	**50.28**	**42.96**	27.14	**18.77**	29.30	24.47	23.95	71.00	86.47	43.48	**3.37**	84.73	55.55	49.30	41.71
ShallowKV + PyramidKV		26.11	29.55	49.41	42.85	27.42	18.36	29.81	**24.55**	24.46	71.00	86.39	42.80	**3.37**	84.62	**56.45**	**50.56**	**41.73**

Table 1: Main LongBench results on Mistral-7B-Instruct-v0.2. Numbers in blue indicate improvements over the corresponding base method, e.g., ShallowKV+H2O vs. H2O. Bold numbers indicate the best results within the same compression ratio.

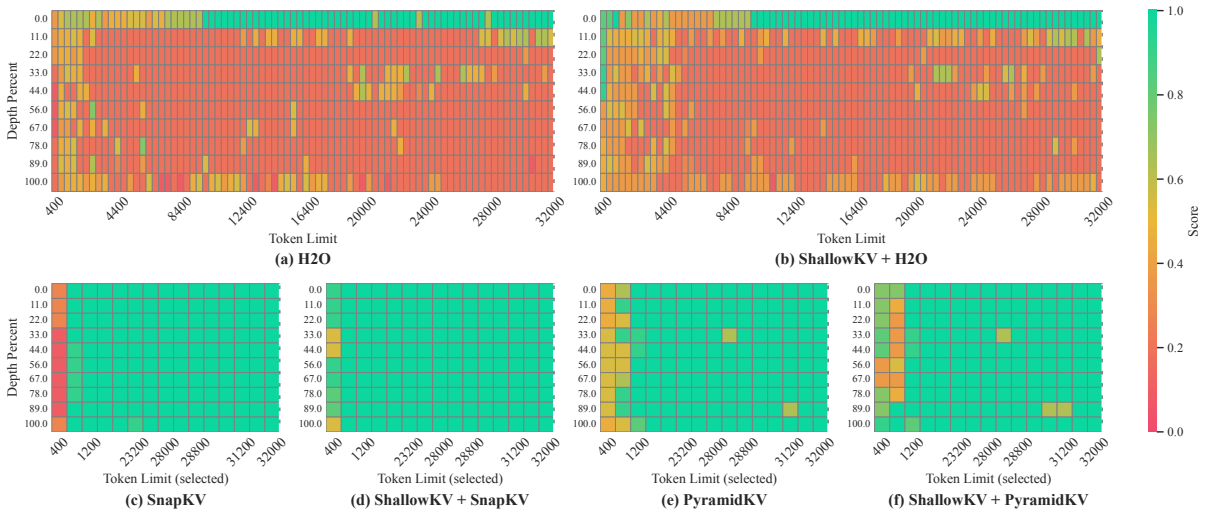


Figure 5: Compact Needle-in-a-Haystack results on Mistral-7B with retention ratio 0.15. The first row shows H2O and its ShallowKV-enhanced variant with the full token range, while the second row summarizes SnapKV, PyramidKV, and their ShallowKV-enhanced variants using selected informative token lengths.

Layer routing. H2O and SnapKV route layers 0–3 to ShallowKV, while PyramidKV routes layers 0–2 to ShallowKV. Energy thresholds are 0.98 for layer 0 and 0.95 for layers 1–3.

Metrics. We report official LongBench task scores, macro averages, decode throughput, and peak memory on an H100 GPU.

4.2 Main Results on Mistral-7B

As shown in Table 1, ShallowKV generally enhances all three retention baselines on Mistral-7B under matched KV budgets. Improvements are most pronounced on tasks requiring broad context aggregation (e.g., HotpotQA, MultiFieldQA). Crucially, more aggressive compression (ratio 0.2) yields larger overall gains, demonstrating the heightened importance of preserving shallow-layer residuals under strict memory constraints.

Conversely, the performance improvements are not uniform across all tasks. Low-rank reconstruction inherently acts as a semantic low-pass filter: while it excels at preserving global contextual information, it can attenuate fine-grained, high-frequency token variations. Consequently, on tasks that strictly rely on exact syntactic matching or precise local details—such as code completion (LCC) or specific few-shot template adherence—the low-rank representation can inadvertently weaken the model's focus on critical discrete tokens. In these scenarios, allocating memory to exact token retention proves more effective than distributed low-rank approximation.

Method	Ratio	Single-Doc		Multi-Doc		Summ.	Few-shot		Synth.	Code	Avg.
		NQA	Qasp.	Hot.	Mus.	Gov.	TREC	SAM.	PCount	LCC	
FullKV	1.0	18.89	21.87	27.83	8.31	27.01	64.00	41.44	3.34	60.51	30.36
H2O	0.4	17.87	18.15	22.36	8.99	**28.49**	63.00	36.70	2.31	49.95	27.54
SnapKV		19.10	24.01	24.35	8.10	24.39	**64.00**	41.07	2.30	60.99	29.81
PyramidKV		18.48	21.19	**31.16**	**10.29**	23.62	**64.00**	41.58	**6.00**	59.68	30.67
ShallowKV + H2O		**19.46**	24.35	25.15	8.43	25.18	**64.00**	40.72	2.80	59.48	29.95
ShallowKV + SnapKV		19.29	24.17	24.02	8.16	24.22	**64.00**	41.29	2.50	61.05	29.86
ShallowKV + PyramidKV		18.99	**24.49**	30.70	9.67	25.12	**64.00**	**41.96**	5.44	**61.15**	**31.28**
H2O	0.2	15.09	14.20	21.52	5.90	**25.56**	62.00	33.62	4.24	40.08	24.69
SnapKV		19.45	21.83	23.80	8.05	22.81	**64.00**	39.94	2.08	59.16	29.01
PyramidKV		17.26	21.08	30.88	**9.57**	21.60	**64.00**	40.31	**6.00**	59.40	30.01
ShallowKV + H2O		17.88	**23.34**	25.15	8.25	24.40	63.00	39.90	2.31	59.26	29.28
ShallowKV + SnapKV		**19.64**	21.93	23.93	8.00	22.40	**64.00**	40.00	2.30	59.56	29.08
ShallowKV + PyramidKV		19.46	22.09	**30.94**	9.16	22.58	**64.00**	**40.83**	5.10	**59.70**	**30.43**

Table 2: LongBench results on Llama-2-7B.

Method	Ratio	Single-Doc		Multi-Doc		Summ.	Few-shot		Synth.	Code	Avg.
		NQA	Qasp.	Hot.	Mus.	Gov.	TREC	SAM.	PCount	LCC	
FullKV	1.0	26.17	12.13	14.18	6.97	32.29	69.00	43.50	2.62	67.79	30.52
H2O	0.4	24.57	11.38	12.80	7.00	**33.55**	69.00	40.37	2.27	61.30	29.14
SnapKV		22.64	**12.91**	13.68	**7.89**	29.88	69.50	**44.22**	**3.20**	67.26	30.13
PyramidKV		26.77	10.96	13.15	6.58	30.13	69.00	43.25	2.67	67.47	30.00
ShallowKV + H2O		25.67	11.42	13.42	7.24	**33.55**	68.00	40.51	2.29	61.76	29.32
ShallowKV + SnapKV		26.13	12.59	**13.77**	7.78	30.16	69.50	43.77	2.57	67.71	**30.44**
ShallowKV + PyramidKV		**26.86**	11.22	13.03	6.98	30.43	**70.00**	43.77	2.63	**67.65**	30.29

Table 3: LongBench results on Llama-3.1-8B.

Model	Wins	Ties	Losses	Win rate
Mistral-7B	65	3	28	69.9%
Llama-2-7B	37	4	13	74.0%
Llama-3.1-8B	18	2	7	72.0%
Overall	120	9	48	71.4%

Table 4: Task-level win-rate of ShallowKV-enhanced variants against their corresponding retention backends. Win rate is computed as Wins/(Wins + Losses).

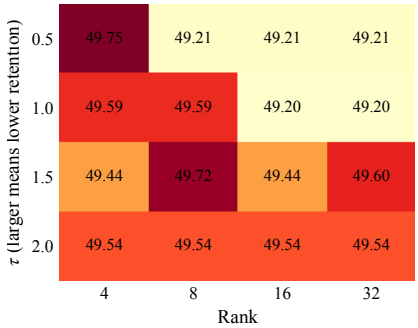


Figure 6: Sensitivity of ShallowKV + SnapKV to rank and τ on MultiFieldQA using Mistral-7B. Cell values denote scores.

4.3 Generalization Across Model Families

Tables 2 and 3 demonstrate ShallowKV's consistent trends across evaluated 7B-8B models, generally enhancing retention backends on both Llama-2-7B and Llama-3.1-8B. The improvements are particularly prominent in tasks requiring long-range evidence aggregation (e.g., Qasper, GovReport). Conversely, occasional task-level degradations occur primarily when the required evidence is highly localized. In such cases, reallocating the strict KV budget from exact token retention to low-rank residual factors dilutes the capacity for capturing precise, local details. Despite this inherent trade-off, ShallowKV achieves a 71.4% overall non-tied win rate. A paired Wilcoxon signed-rank test confirms the statistical significance of these gains across Mistral-7B, Llama-2-7B, and Llama-3.1-8B ($p = 6.66 \times 10^{-6}, 1.33 \times 10^{-3}$, and 4.95×10^{-2}, respectively), yielding a highly significant $p = 6.15 \times 10^{-9}$ overall.

4.4 Component Ablation

Table 5 dissects the contribution of each ShallowKV component on top of the SnapKV baseline. Progressively integrating prefill-stage SVD, decode-side recompression, and adaptive rank selection yields steady performance gains, confirming the necessity of dynamic low-rank residual preservation. Notably, the introduction of semantic chunking and representative-query scoring provides the most substantial improvements (+0.32 and +0.23 at ratio 0.4, respectively). This indicates that aligning compression boundaries with natural text structures and augmenting prefill scoring with prompt-wide queries effectively prevents the loss of global context, resolving the short-sightedness of purely local window-based scoring.

4.5 Layer Routing Analysis

Figure 7 studies how many shallow layers should be routed to ShallowKV. We vary the routed range from no ShallowKV layers to layers 0, 0–1, 0–3, 0–7, and 0–11 on top of H2O at ratio 0.4.

The non-monotonic trend reflects a fidelity-budget trade-off. Shallow-layer KV states have stronger low-rank structure, so ShallowKV preserves distributed contextual evidence efficiently. Deeper-layer KV states require higher ranks to reach the same spectral energy, so routing too many layers to ShallowKV spends more budget on low-rank factors, leaving less budget for the retention backend. This causes task-level fluctuations, with the best overall balance around layers 0–3.

4.6 Sensitivity to Rank and Exact-Anchor Threshold

Figure 6 reports the sensitivity of ShallowKV + SnapKV to rank and the exact-anchor threshold τ on MultiFieldQA. ShallowKV remains stable across a broad range of settings, suggesting that it is not tied to a narrow hyperparameter choice.

Method	Ratio	MFQA	HotpotQA	GovReport	PCount	Avg.
SnapKV baseline	0.4	49.47	42.98	31.01	2.42	31.47
ShallowKV-prefill + SnapKV		49.45	42.68	31.23	2.70	31.52
+ Decode recompression.		49.50	42.93	31.36	2.85	31.66
+ Adaptive rank		49.71	43.47	31.45	2.86	31.87
+ Semantic chunking		49.66	43.49	31.49	3.00	31.91
+ Rep. queries		**49.73**	**44.01**	**31.68**	**3.13**	**32.14**
SnapKV baseline	0.3	49.41	42.69	30.14	2.55	31.20
ShallowKV-prefill + SnapKV		49.73	42.66	30.35	2.85	31.40
+ Decode recompression.		49.78	42.45	30.41	2.92	31.39
+ Adaptive rank		49.74	42.90	30.43	3.11	31.55
+ Semantic chunking		**49.92**	43.22	30.81	3.14	31.77
+ Rep. queries		**49.92**	**43.26**	**30.94**	**3.27**	**31.85**
SnapKV baseline	0.2	49.19	41.97	29.48	2.39	30.76
ShallowKV-prefill + SnapKV		49.31	41.32	**29.77**	2.90	30.83
+ Decode recompression.		49.50	42.31	29.57	2.96	31.09
+ Adaptive rank		50.26	41.99	29.11	3.25	31.15
+ Semantic chunking		50.20	42.82	28.99	3.21	31.31
+ Rep. queries		**50.28**	**42.96**	29.30	**3.37**	**31.48**

Table 5: Component ablation of ShallowKV on top of SnapKV under different retention ratios. Starting from the SnapKV baseline, we progressively add prefill-stage SVD compression, decode-side recompression, adaptive rank selection, semantic chunking, and representative-query scoring. The final row corresponds to the full ShallowKV + SnapKV method.

Model	FullKV	+H2O	+SnapKV	+PyramidKV
Mistral-7B	1.00	0.3977	0.4065	0.3995
Llama-2-7B	1.00	0.3998	0.4084	0.3996
Llama-3.1-8B	1.00	0.4015	0.4015	0.4059

Table 6: Peak memory normalized by FullKV.

The best score appears at rank 4 and $\tau = 0.5$, indicating that shallow-layer residuals can be represented with small low-rank factors and that, after sufficient residual preservation, exact anchors become more useful than further increasing the rank.

4.7 Needle-in-a-Haystack Evaluation

As shown in Figure 5, ShallowKV improves retrieval robustness across backends. It mitigates H2O's broad failures and the localized near-side weaknesses of SnapKV and PyramidKV, demonstrating that low-rank residuals effectively recover evidence missed by pure retention.

4.8 Memory–Throughput Trade-off

Tables 6 and 7 show that all ShallowKV variants (+H2O/SnapKV/PyramidKV) reduce peak memory to ∼40% of FullKV. ShallowKV evaluates historical chunks online from their SVD factors on GPU, making speed architecture-dependent. On GQA models, fewer key-value heads are stored and read than query heads, so FullKV already has low KV-cache traffic and ShallowKV's traffic savings are smaller relative to its extra SVD computation. On the MHA Llama-2-7B model, KV-cache traffic is higher, so ShallowKV reduces historical KV movement and achieves faster decoding than FullKV.

Model	FullKV	+H2O	+SnapKV	+PyramidKV
Mistral-7B	20.06	12.56	14.29	14.16
Llama-2-7B	14.95	20.78	21.24	21.33
Llama-3.1-8B	22.40	18.28	20.36	20.18

Table 7: Decode speed on an H100 GPU (tokens/s).

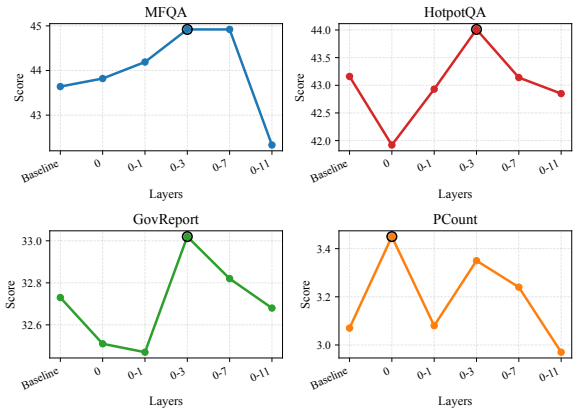


Figure 7: Layer-range ablation for ShallowKV + H2O on Mistral-7B at retention ratio 0.4.

5 Conclusion

Across LongBench, cross-model experiments, ablations, and retrieval stress tests, ShallowKV generally improves strong retention-based compressors under the same KV budget. The gains are most evident when the memory budget is aggressive and when tasks require preserving information from broad prompt regions. These results support ShallowKV's core design: shallow layers benefit from low-rank residual preservation, while deeper layers are better handled by attention-based retention.

6 Limitations

We identify two primary limitations. First, prefill SVD and scoring overheads limit end-to-end speedups without custom hardware-aware kernels. Second, low-rank residual preservation inherently acts as a semantic low-pass filter: while effective for broad-context aggregation, it attenuates fine-grained token variations. Consequently, ShallowKV may underperform on detail-sensitive tasks requiring exact syntactic matching, such as code completion (e.g., LCC). Future work will explore task-adaptive layer routing to mitigate this trade-off.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: KV cache reduction through key tokens selection for efficient generative inference. In *Proceedings of Machine Learning and Systems*.

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

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

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing GPT-4 with 90% chatgpt quality.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Yuhui Fu, Zongwei Cai, Ali Asi, Wei Xiong, Yuxin Dong, and Wenji Xiao. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations*.

Ziwei He, Jian Yuan, Haoli Bai, Jingwen Leng, and Bo Jiang. 2025. Treekv: Smooth key-value cache compression with tree structures. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*.

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. In *Advances in Neural Information Processing Systems*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Minsu Kim, Seongmin Hong, RyeoWook Ko, Soongyu Choi, Hunjong Lee, Junsoo Kim, Joo-Young Kim, and Jongse Park. 2025. Oaken: Fast and efficient llm serving with online-offline hybrid kv cache quantization. *Preprint*, arXiv:2503.18599.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Jia Li, Ge Li, Yongmin Li, and Zhi-Qiang Jin. 2025. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2):37:1–37:23.

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. *Preprint*, arXiv:2404.14469.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2025. Qserve: W4a8kv4 quantization and system co-design for efficient LLM serving. In *Proceedings of Machine Learning and Systems*.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *arXiv preprint arXiv:2502.00299*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for KV cache. *Preprint*, arXiv:2402.02750.

Da Ma, Lu Chen, Situo Zhang, Yuxun Miao, Su Zhu, Zhi Chen, Hongshen Xu, Hanqi Li, Shuai Fan, Lei Pan, and Kai Yu. 2024. Compressing kv cache for long-context llm inference with inter-layer attention similarity. *Preprint*, arXiv:2412.02252.

Meta AI. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

NVIDIA. 2023. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, volume 5.

Utkarsh Saxena and Kaushik Roy. 2025. Kvlinc: Kv cache quantization with hadamard rotation and linear correction. *Preprint*, arXiv:2510.05373.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. 2024. Loki: Low-rank keys for efficient sparse attention. In *Advances in Neural Information Processing Systems*.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2024. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *Preprint*, arXiv:2410.21465.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2024. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Hao Wang, Ligong Han, Kai Xu, and Akash Srivastava. 2025. Squat: Subspace-orthogonal kv cache quantization. *Preprint*, arXiv:2503.24358.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. 2025. Scope: Optimizing key-value cache compression in long-context generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*.

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

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. Pyramidinfer: Pyramid KV cache compression for high-throughput llm inference. *Preprint*, arXiv:2405.12532.

Bohan Yu and Yekun Chai. 2025. Evolkv: Evolutionary kv cache compression for llm inference. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Amir Zandieh, Majid Daliri, and Insu Han. 2025. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. 2024. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *Preprint*, arXiv:2308.07107.

A Additional Results

A.1 Layer Routing

Table 8 analyzes how the choice of routed shallow layers affects performance. The results support routing only a small number of early layers to ShallowKV.

B Implementation Details

This section summarizes the implementation settings used in our experiments. We include layer routing, backend retention ratios, ShallowKV-specific hyperparameters, and memory accounting rules.

B.1 Dataset Abbreviations

Table 9 lists the dataset abbreviations used in the LongBench result tables.

B.2 Hyperparameters

B.3 Memory Accounting

Counted components. For budget-matched comparisons, the KV budget includes exact anchor keys and values, low-rank key and value factors, sink-token keys and values, dense local-window keys and values, decode-buffer keys, queries, and values, and recompressed decode chunks. We exclude only negligible bookkeeping metadata, such as token indices and chunk boundaries, since these are small compared with KV tensors.

C Algorithms

This section provides procedural details for the ShallowKV pipeline described in Section 3. Algorithm 1 first summarizes the semantic-boundary chunking procedure used to construct the prefill chunks. Algorithm 2 then instantiates the prefill-stage compression in shallow routed layers, following the exact-anchor and adaptive low-rank residual design introduced in Section 3.2. Algorithm 3 summarizes the decode-side recompression process described in Section 3.3. Finally, algorithm 4 describes the CUDA decode workflow, whose mathematical equivalence to attention over reconstructed KV states is shown in Section C.5.

C.1 Semantic-Boundary Chunking

Algorithm 1 summarizes the chunking procedure used by ShallowKV. The procedure is a lightweight heuristic rather than a separate semantic chunking model. It first builds short base blocks, aligns their ends to nearby natural boundaries, scores candidate boundaries using adjacent block discontinuity, and finally selects chunk boundaries under minimum, target, and maximum length constraints.

Algorithm 1 Semantic-Boundary Chunking

Require: token pieces $\{x_i\}_{i=1}^{T}$, key states $\mathbf{K}$
Require: minimum, target, and maximum chunk sizes m, t, M
Require: base block size b and alignment window w
Ensure: chunk spans $\mathcal{C}$

// Build structure-aligned base blocks.
$\mathcal{B} \leftarrow \emptyset,\ s \leftarrow 1$
while $s \leq T$ **do**
  $\bar{e} \leftarrow \min(T, s + b - 1)$
  $\Omega \leftarrow [\bar{e} - w, \bar{e} + w] \cap [s, T]$
  // Protected filters invalid cuts, e.g., unfinished brackets, signatures before bodies, or incomplete control blocks;
  // Nat rewards natural breaks such as punctuation and blank lines.
  $\Omega_v \leftarrow \{e \in \Omega : \neg\text{Protected}(e)\}$
  $e^\star \leftarrow \arg\max_{e \in \Omega_v} [\text{Nat}(e) - \lambda|\bar{e} - e|]$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{[s, e^\star]\};\quad s \leftarrow e^\star + 1$
end while
// Score semantic shifts between neighboring blocks.
$\mathbf{h}_j \leftarrow \text{Normalize}\big(\text{Mean}(\mathbf{K}_{B_j})\big)$ for each block $B_j \in \mathcal{B}$
$S(e_j) \leftarrow 1 - \cos(\mathbf{h}_j, \mathbf{h}_{j+1}) + \alpha\,\text{Nat}(e_j)$
  for each adjacent-block boundary e_j
// Assemble chunks under the size constraints.
$\mathcal{C} \leftarrow \emptyset,\ s \leftarrow 1$
while $s \leq T$ **do**
  $\Gamma_s \leftarrow \{e_j : m \leq e_j - s + 1 \leq M\}$
  if $\Gamma_s \neq \emptyset$ **then**
    $e^\star \leftarrow \arg\max_{e \in \Gamma_s} [S(e) - \beta|(e - s + 1) - t|]$
  else
    $\Omega_s \leftarrow [s + m - 1, \min(T, s + M - 1)]$
    $e^\star \leftarrow \arg\max_{e \in \Omega_s} [\text{Nat}(e) - \beta|(e - s + 1) - t|]$
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{[s, e^\star]\};\quad s \leftarrow e^\star + 1$
end while
return $\mathcal{C}$

C.2 Prefill Compression

During prefill, ShallowKV builds the initial compressed cache for shallow routed layers. Consistent with the formulation in Eq. 1, layers routed to the retention backend are compressed by the corresponding retention method, while ShallowKV layers preserve exact anchors and compress the residual states with adaptive-rank SVD **(detailed in Algorithm 2)**.

C.3 Decode Compression

During decoding, ShallowKV keeps newly generated states in a dense recent window and periodically recompresses older generated states after they

Backend	Ratio	Routed ShallowKV Layers	MultiFieldQA	HotpotQA	GovReport	Passage Count
H2O	0.4	Baseline	43.64	43.16	32.73	3.07
H2O	0.4	0	43.82	41.92	32.51	**3.45**
H2O	0.4	0–1	44.19	42.93	32.47	3.08
H2O	0.4	0–3	**44.92**	**44.01**	**33.02**	3.35
H2O	0.4	0–7	**44.92**	43.14	32.82	3.24
H2O	0.4	0–11	42.33	42.85	32.68	2.97
H2O	0.2	Baseline	39.75	41.02	31.89	2.86
H2O	0.2	0	40.84	40.46	31.61	2.79
H2O	0.2	0–3	**40.93**	**41.74**	**32.03**	**3.24**
H2O	0.2	0–7	29.12	35.43	27.86	2.05
SnapKV	0.4	Baseline	49.47	42.98	31.01	2.42
SnapKV	0.4	0	49.60	43.47	31.02	2.55
SnapKV	0.4	0–3	**49.73**	**44.01**	**31.68**	**3.13**
SnapKV	0.4	0–7	49.68	43.21	31.35	2.70

Table 8: Complete layer-routing ablation results for ShallowKV in the appendix. We vary the shallow layers routed to ShallowKV while keeping the remaining layers on the corresponding retention backend. "Baseline" denotes the original backend without ShallowKV.

Abbreviation	Full dataset name
NQA	NarrativeQA
Qasper / Qasp.	Qasper
MFQA	MultiFieldQA-en
Hotpot / Hot.	HotpotQA
2Wiki	2WikiMultihopQA
Musique / Mus.	MuSiQue
Gov.	GovReport
QMSum	QMSum
MNews	MultiNews
TREC	TREC
Trivia	TriviaQA
SAMSum / SAM.	SAMSum
PCount	PassageCount
PR	PassageRetrieval-en
LCC	LCC
Repo	RepoBench-P

Table 9: Dataset abbreviations used in the LongBench result tables.

accumulate into a decode buffer. This procedure implements the local-window-guided recompression rule described in Section 3.3 **(summarized in Algorithm 3)**.

C.4 Chunk-wise CUDA Decode Attention

Algorithm 4 summarizes the CUDA decode path used by ShallowKV. The implementation avoids reconstructing the full historical KV cache. Instead, it computes attention statistics directly from exact retained tokens and SVD factors, and then merges them with the dense recent-window state using online softmax.

C.5 Equivalence of Chunk-wise CUDA Attention

We show that the chunk-wise CUDA implementation is mathematically equivalent to attention over the explicitly reconstructed KV cache, up to floating-point roundoff. The key point is that the implementation does not materialize the full reconstructed historical cache. Instead, it computes the same attention scores and value-weighted sums directly from exact tokens and SVD factors.

RoPE-aware scores for SVD keys. For a compressed token i in a chunk, ShallowKV stores its key as

$$\mathbf{k}_i = \sum_{j=1}^{r} \lambda_{i,j}^{K} \mathbf{b}_j^{K}, \tag{14}$$

where $b_j^K \in \mathbb{R}^d$ is the j-th key basis vector and $\lambda_{i,j}^K$ is the token-specific coefficient. With rotary position embedding, the standard attention score is

$$s_i = \tau \mathbf{q}^\top \mathrm{RoPE}_{p_i}(\mathbf{k}_i), \quad \tau = \frac{1}{\sqrt{d}}, \tag{15}$$

where p_i is the token position.

Since RoPE is a block-diagonal linear rotation, it distributes over the low-rank reconstruction:

$$\mathrm{RoPE}_{p_i}(k_i) = \mathrm{RoPE}_{p_i}\left(\sum_{j=1}^{r} \lambda_{i,j}^{K} b_j^{K}\right) \tag{16}$$

$$= \sum_{j=1}^{r} \lambda_{i,j}^{K} \mathrm{RoPE}_{p_i}(b_j^{K}). \tag{17}$$

Model	Method	Retention layers	ShallowKV layers	Backend ratio
Mistral-7B	H2O / SnapKV / PyramidKV	0–31	–	0.4 / 0.2
Mistral-7B	ShallowKV + H2O / SnapKV	4–31	0–3	0.38 / 0.16
Mistral-7B	ShallowKV + PyramidKV	3–31	0–2	0.38 / 0.16
Llama-2-7B	H2O / SnapKV / PyramidKV	0–31	–	0.4 / 0.2
Llama-2-7B	ShallowKV + H2O / SnapKV	4–31	0–3	0.38 / 0.16
Llama-2-7B	ShallowKV + PyramidKV	3–31	0–2	0.38 / 0.16
Llama-3.1-8B	H2O / SnapKV / PyramidKV	0–31	–	0.4
Llama-3.1-8B	ShallowKV + H2O / SnapKV	4–31	0–3	0.38
Llama-3.1-8B	ShallowKV + PyramidKV	3–31	0–2	0.38

Table 10: Layer routing and backend retention ratios used in our experiments. Retention-only baselines apply the corresponding backend to all layers. ShallowKV variants route shallow layers to ShallowKV and apply the backend retention method to the remaining layers. The backend ratio is slightly reduced in ShallowKV variants to match the overall KV memory budget after accounting for exact anchors and low-rank factors.

Hyperparameter	Value
Attention implementation	SDPA
Semantic chunking	enabled
Semantic chunk size range	96–256, target 160
Sink tokens	4
Adaptive rank	enabled
Rank range	1–64
Energy threshold	layer 0:0.98; layers 1–3: 0.95
Exact-token mode	subspan-prioritized
Exact span size	4
Outlier top-k	16
Score reduction	mean over queries and heads
Representative queries	enabled
Representative block size	64
Representatives per block	4
Backend local window	64
Decode recompression trigger	512 tokens

Table 11: Main ShallowKV implementation hyperparameters. We report active method-level settings used in the main experiments and omit unused implementation defaults for clarity.

Therefore,

$$s_i = \tau \sum_{j=1}^{r} \lambda_{i,j}^K q^\top \mathrm{RoPE}_{p_i}(b_j^K). \tag{18}$$

Thus, it is sufficient to compute the rotated dot product between the query and each basis vector, then combine the results with the SVD coefficients.

A subtlety is that different SVD tokens in the same chunk have different positions p_i, so the rotated basis $\mathrm{RoPE}_{p_i}(b_j^K)$ cannot be precomputed once and reused for all tokens. However, RoPE rotates each two-dimensional coordinate pair independently, which exposes a reusable structure. Following the split-half RoPE layout used by LLaMA/Mistral models, we take each RoPE pair as $(t, t + d/2)$ and define

$$A_{j,t} = q_t b_{j,t}^K + q_{t+d/2} b_{j,t+d/2}^K, \tag{19}$$

$$B_{j,t} = q_{t+d/2} b_{j,t}^K - q_t b_{j,t+d/2}^K. \tag{20}$$

These two terms depend only on the query q and the basis vector b_j^K, not on the token position. The RoPE-rotated dot product can then be written as

$$\begin{aligned} g_{i,j} &= q^\top \mathrm{RoPE}_{p_i}(b_j^K) \\ &= \sum_t \left(A_{j,t} \cos(p_i \omega_t) + B_{j,t} \sin(p_i \omega_t) \right). \end{aligned} \tag{21}$$

Therefore, the CUDA kernel precomputes the position-independent query-basis products $(A_{j,t}, B_{j,t})$ once for each query, head, and rank basis, and injects each token's position only through $\cos(p_i \omega_t)$ and $\sin(p_i \omega_t)$ to obtain $g_{i,j}$. Combining these basis scores with the token-specific coefficients gives

$$s_i = \tau \sum_{j=1}^{r} \lambda_{i,j}^K g_{i,j}. \tag{22}$$

This is exactly the score obtained by explicitly reconstructing k_i, applying RoPE at position p_i, and computing $q^\top \mathrm{RoPE}_{p_i}(k_i)$.

Exact retained keys are simpler: they are packed after applying RoPE, so the kernel directly computes $s_i = \tau q^\top K_{e,i}$ for exact token i.

Value accumulation. Values do not use rotary position embedding. For an SVD-compressed token i, the reconstructed value is

$$\mathbf{v}_i = \sum_{j=1}^{r} \lambda_{i,j}^V \mathbf{b}_j^V. \tag{23}$$

The kernel computes this value contribution on the fly from $(\lambda_{i,j}^V, b_j^V)$ and immediately accumulates it into the attention numerator. Thus the full reconstructed value cache is never materialized, but the value contribution is the same as using the reconstructed v_i.

Online softmax equivalence. It remains to show that chunk-wise accumulation gives the same result as a global softmax over all dense, exact, and SVD-compressed tokens. For any token set $\mathcal{S}$, define the online softmax state

$$m_{\mathcal{S}} = \max_{i \in \mathcal{S}} s_i, \tag{24}$$

$$z_{\mathcal{S}} = \sum_{i \in \mathcal{S}} \exp(s_i - m_{\mathcal{S}}), \tag{25}$$

$$u_{\mathcal{S}} = \sum_{i \in \mathcal{S}} \exp(s_i - m_{\mathcal{S}}) v_i. \tag{26}$$

The attention output over $\mathcal{S}$ is

$$o_{\mathcal{S}} = \frac{u_{\mathcal{S}}}{z_{\mathcal{S}}}. \tag{27}$$

Consider two disjoint token sets $\mathcal{A}$ and $\mathcal{B}$ with states $(m_{\mathcal{A}}, z_{\mathcal{A}}, u_{\mathcal{A}})$ and $(m_{\mathcal{B}}, z_{\mathcal{B}}, u_{\mathcal{B}})$. Let

$$m = \max(m_{\mathcal{A}}, m_{\mathcal{B}}). \tag{28}$$

Define the rescaling factors as

$$\alpha_{\mathcal{A}} = \exp(m_{\mathcal{A}} - m), \tag{29}$$

$$\alpha_{\mathcal{B}} = \exp(m_{\mathcal{B}} - m). \tag{30}$$

The merged state is then computed as

$$z = z_{\mathcal{A}} \alpha_{\mathcal{A}} + z_{\mathcal{B}} \alpha_{\mathcal{B}}, \tag{31}$$

$$u = u_{\mathcal{A}} \alpha_{\mathcal{A}} + u_{\mathcal{B}} \alpha_{\mathcal{B}}. \tag{32}$$

Substituting the definitions of z and u gives

$$z = \sum_{i \in \mathcal{A} \cup \mathcal{B}} \exp(s_i - m), \tag{33}$$

$$u = \sum_{i \in \mathcal{A} \cup \mathcal{B}} \exp(s_i - m) v_i. \tag{34}$$

Hence u/z is exactly the same output as applying softmax once over $\mathcal{A} \cup \mathcal{B}$. By repeatedly applying this merge rule, the CUDA kernel combines dense recent tokens, exact retained tokens, and all SVD-compressed chunks into the same attention result as global attention over the fully reconstructed KV cache.

Algorithm 2 ShallowKV Prefill Compression

Require: prompt states $\{\mathbf{K}^{(l)}, \mathbf{Q}^{(l)}, \mathbf{V}^{(l)}\}_{l=0}^{L-1}$
Require: ShallowKV layers $\mathcal{L}_{\text{svd}}$, retention layers $\mathcal{L}_{\text{ret}}$
Require: exact threshold coefficient τ, representative count m, smoothing kernel k_{peak}
Ensure: compressed prompt cache $\mathcal{M} = \{M^{(l)}\}_{l=0}^{L-1}$

for $l = 0$ to $L-1$ **do**
  if $l \in \mathcal{L}_{\text{ret}}$ **then**
    $M^{(l)} \leftarrow \text{Retain}(\mathbf{K}^{(l)}, \mathbf{V}^{(l)})$
  else
    split $(\mathbf{K}^{(l)}, \mathbf{Q}^{(l)}, \mathbf{V}^{(l)})$ into sink states, middle states, and local window
    divide middle states into semantic-boundary chunks $\mathcal{C}$
    $\mathbf{Q}_{\text{tail}} \leftarrow$ queries from the prompt suffix
    $\mathcal{W}_{\text{rep}} \leftarrow \text{CoarseSelect}(\mathbf{Q}_{\text{tail}}, \mathbf{Q}^{(l)}, \mathcal{C})$
    $\mathbf{Q}_{\text{rep}} \leftarrow \text{FineSelect}(\mathbf{Q}_{\text{tail}}, \mathbf{K}_{\mathcal{W}_{\text{rep}}})$
    for chunk $c \in \mathcal{C}$ **do**
      $\mathbf{Q}_{\text{score}}(c) \leftarrow [\mathbf{Q}_{\text{tail}}; \mathbf{Q}_{\text{rep}}]$
      $\mathbf{a}_c \leftarrow \text{Score}(\mathbf{Q}_{\text{score}}(c), \mathbf{K}_c)$
      $\mathcal{E}_c \leftarrow \text{ExactSelect}(\mathbf{a}_c)$
      $\mathcal{R}_c \leftarrow c \setminus \mathcal{E}_c$
      $\mathbf{E}_c \leftarrow (\mathbf{K}_{\mathcal{E}_c}, \mathbf{V}_{\mathcal{E}_c})$
      $r_c \leftarrow \text{EnergyRank}(\mathbf{K}_{\mathcal{R}_c}, \mathbf{V}_{\mathcal{R}_c}; \eta_l)$
      $(\hat{\mathbf{K}}_{\mathcal{R}_c}, \hat{\mathbf{V}}_{\mathcal{R}_c}) \leftarrow$
        $\text{LowRankCompress}(\mathbf{K}_{\mathcal{R}_c}, \mathbf{V}_{\mathcal{R}_c}; r_c)$
      store $\{\mathbf{E}_c, \hat{\mathbf{K}}_{\mathcal{R}_c}, \hat{\mathbf{V}}_{\mathcal{R}_c}\}$ in $M^{(l)}$
    end for
    append sink states and local window to $M^{(l)}$
  end if
end for
return $\mathcal{M}$

ExactSelect implementation.
Given token scores $\mathbf{a}_c = \{a_i\}_{i \in c}$ for chunk c:
  $B_c \leftarrow |\{i \in c : a_i > \text{mean}(\mathbf{a}_c) + \tau\, \text{std}(\mathbf{a}_c)\}|$
  $\tilde{\mathbf{a}}_c \leftarrow \text{AvgPool1D}(\mathbf{a}_c; k_{\text{peak}})$
  $\mathcal{P}_c \leftarrow \emptyset$, $\rho \leftarrow \max(1, \lfloor |c|/(2m) \rfloor)$
  while $|\mathcal{P}_c| < m$ **and** $\tilde{\mathbf{a}}_c$ has finite entries **do**
    $p^\star \leftarrow \arg\max_{i \in c} \tilde{a}_i$
    $\mathcal{P}_c \leftarrow \mathcal{P}_c \cup \{p^\star\}$
    Set $\tilde{a}_j \leftarrow -\infty$ for all j with $|j - p^\star| \le \rho$
  end while
  If $|\mathcal{P}_c| < m$, add the remaining highest-scoring positions not in $\mathcal{P}_c$.
  Sort $\mathcal{P}_c$ by position.
  Construct anchor-induced subspans $\mathcal{S}_c$ by placing boundaries at
  midpoints between adjacent positions in $\mathcal{P}_c$.
  $s^\star \leftarrow \arg\max_{s \in \mathcal{S}_c} \sum_{i \in s} a_i$
  $\mathcal{E}_c \leftarrow \text{TopK}(\{a_i : i \in s^\star\}, B_c)$
  If $|\mathcal{E}_c| < B_c$, fill the remaining budget using top-scoring tokens in
  $c \setminus s^\star$.

Algorithm 3 ShallowKV Decode Compression

Require: cache $\mathcal{M} = \{M^{(l)}\}_{l=0}^{L-1}$, recent windows $\{\mathbf{W}^{(l)}\}_{l=0}^{L-1}$
Require: decode buffers $\{\mathcal{B}^{(l)}\}_{l=0}^{L-1}$
Require: new-step states $\{\mathbf{k}_t^{(l)}, \mathbf{q}_t^{(l)}, \mathbf{v}_t^{(l)}\}_{l=0}^{L-1}$
Ensure: updated cache $\mathcal{M}$, windows $\{\mathbf{W}^{(l)}\}$, buffers $\{\mathcal{B}^{(l)}\}$

for $l = 0$ to $L-1$ **do**
  if $l \in \mathcal{L}_{\text{ret}}$ **then**
    $M^{(l)} \leftarrow \text{RetentionUpdate}(M^{(l)}, \mathbf{k}_t^{(l)}, \mathbf{v}_t^{(l)})$
  else
    append $(\mathbf{q}_t^{(l)}, \mathbf{k}_t^{(l)}, \mathbf{v}_t^{(l)})$ to $\mathbf{W}^{(l)}$
    if $|\mathbf{W}^{(l)}| > w$ **then**
      move the oldest state from $\mathbf{W}^{(l)}$ to $\mathcal{B}^{(l)}$
    end if
    if $|\mathcal{B}^{(l)}| \geq m$ **then**
      $\mathcal{C}_{\text{old}}^{(l)} \leftarrow \text{OldestChunk}(\mathcal{B}^{(l)})$
      $\mathbf{Q}_{\text{win}}^{(l)} \leftarrow \text{Queries}(\mathbf{W}^{(l)})$
      $\mathbf{a}^{(l)} \leftarrow \text{Score}(\mathbf{Q}_{\text{win}}^{(l)}, \mathbf{K}_{\mathcal{C}_{\text{old}}^{(l)}})$
      $\mathcal{E}^{(l)} \leftarrow \text{ExactSelect}(\mathbf{a}^{(l)})$
      $\mathcal{R}^{(l)} \leftarrow \mathcal{C}_{\text{old}}^{(l)} \setminus \mathcal{E}^{(l)}$
      $\mathbf{E}^{(l)} \leftarrow (\mathbf{K}_{\mathcal{E}^{(l)}}, \mathbf{V}_{\mathcal{E}^{(l)}})$
      $r^{(l)} \leftarrow \text{EnergyRank}(\mathbf{K}_{\mathcal{R}^{(l)}}, \mathbf{V}_{\mathcal{R}^{(l)}}; \eta_l)$
      $(\hat{\mathbf{K}}_{\mathcal{R}^{(l)}}, \hat{\mathbf{V}}_{\mathcal{R}^{(l)}}) \leftarrow$
        $\text{LowRankCompress}(\mathbf{K}_{\mathcal{R}^{(l)}}, \mathbf{V}_{\mathcal{R}^{(l)}}; r^{(l)})$
      $\text{StoreDecodeChunk}(M^{(l)}, \mathbf{E}^{(l)}, \hat{\mathbf{K}}_{\mathcal{R}^{(l)}}, \hat{\mathbf{V}}_{\mathcal{R}^{(l)}})$
      remove $\mathcal{C}_{\text{old}}^{(l)}$ from $\mathcal{B}^{(l)}$
    end if
  end if
end for
return $\mathcal{M}, \{\mathbf{W}^{(l)}\}_{l=0}^{L-1}, \{\mathcal{B}^{(l)}\}_{l=0}^{L-1}$

Algorithm 4 ShallowKV Chunk-wise CUDA Decode Attention

Require: RoPE-applied decode query q
Require: dense recent/sink state (m_d, z_d, u_d)
Require: exact values V_e and RoPE-applied exact keys K_e
Require: SVD factors $(B^K, B^V, \Lambda^K, \Lambda^V)$, positions p, and chunk pointers
Require: RoPE inverse frequencies ω, scale $\tau = 1/\sqrt{d}$
Ensure: decode attention output o

Launch one CUDA block for each compressed chunk c and query head h.
for each chunk c and head h in parallel **do**
  Load query q_h into shared memory.
  Initialize chunk online-softmax state:
    $m_c \leftarrow -\infty, \quad z_c \leftarrow 0, \quad u_c \leftarrow \mathbf{0}.$
  Load the SVD rank range of chunk c and let its rank be r_c.
  for each rank basis $j = 1, \ldots, r_c$ and RoPE pair t in parallel **do**
    Precompute position-independent query-basis products:
      $A_{j,t} \leftarrow q_{h,t} B_{j,t}^K + q_{h,t+d/2} B_{j,t+d/2}^K$
      $B_{j,t}^{\text{rope}} \leftarrow q_{h,t+d/2} B_{j,t}^K - q_{h,t} B_{j,t+d/2}^K$
  end for
  for each exact token i in chunk c **do**
    Compute exact-key score:
      $s_i \leftarrow \tau q_h^\top K_{e,i}$
    Update online-softmax state with $(s_i, V_{e,i})$:
      $(m_c, z_c, u_c) \leftarrow \text{OnlineUpdate}(m_c, z_c, u_c, s_i, V_{e,i})$
  end for
  for each SVD-compressed token i in chunk c **do**
    Recover its RoPE-aware key score without materializing k_i:
      $s_i \leftarrow \tau \sum_{j=1}^{r_c} \lambda_{i,j}^K \sum_t \left(A_{j,t} \cos(p_i \omega_t) + B_{j,t}^{\text{rope}} \sin(p_i \omega_t)\right)$
    Recover its value on the fly:
      $v_i \leftarrow \sum_{j=1}^{r_c} \lambda_{i,j}^V B_j^V$
    Update online-softmax state with (s_i, v_i):
      $(m_c, z_c, u_c) \leftarrow \text{OnlineUpdate}(m_c, z_c, u_c, s_i, v_i)$
  end for
  Store chunk state (m_c, z_c, u_c).
end for
Merge all compressed-chunk states over chunks:
  $(m_h, z_h, u_h) \leftarrow \text{OnlineMerge}(\{(m_c, z_c, u_c)\}_c)$.
Merge historical compressed state with dense recent/sink state:
  $(m, z, u) \leftarrow \text{OnlineMerge}((m_h, z_h, u_h), (m_d, z_d, u_d))$.
return $o \leftarrow u/z$.