S^{4}R: Selective Sampling, Subspaces, and Sparse Reconstruction for Compressed Long-Context KV Caching

Jialong Han, You Wu, Kewei Tu*
School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
{hanjl2026, wuyou2024, tukw}@shanghaitech.edu.cn

Abstract

The growth of context window lengths in Large Language Models (LLMs) significantly enhances their long-context capabilities but incurs prohibitive memory costs due to the Key-Value (KV) cache. Although low-rank compression of KV cache is a promising remedy, existing methods face a dilemma: offline approaches depend on external calibration data, whereas online approaches incur substantial compute for full-prompt decomposition and reconstruction. In this paper, we propose **S^{4}R**, which builds low-rank subspaces from selectively sampled tokens and computes attention over a sparsely reconstructed KV representation. S^{4}R uses prompt-aware initialization to build initial key/value bases from a representative prompt subset, trading off calibration-data dependence against prefilling cost. Because fully reconstructing the cache at every decoding step is prohibitively expensive and hurts throughput, we further adopt sparse reconstruction to retain only informative positions during decoding. Extensive experiments on LongBench and RULER with Llama and Qwen model families show that S^{4}R achieves up to **5**$\times$ KV compression with near full-cache accuracy, combining the efficiency of fixed compression with the adaptability of prompt-dependent methods. Our code is publicly available at https://github.com/Draconis98/S4R

1 Introduction

The remarkable expansion of context window sizes in Large Language Models (LLMs) — now spanning from 128k up to millions of tokens (OpenAI et al., 2024; Team et al., 2024; Yang et al., 2025b) — has ushered in a new era of capabilities, powering breakthroughs in long-document summarization (Bai et al., 2025), retrieval-augmented generation (Lewis et al., 2020), and autonomous agents (Park et al., 2023; Wang et al., 2024a). The scaling of context window brings an acute bottleneck: memory consumption. As context length increases, the Key-Value (KV) cache — used for storing intermediate computations — can rapidly surpass the footprint of the model parameters themselves (Kwon et al., 2023; Pope et al., 2023; Ainslie et al., 2023), making memory efficiency the central challenge in LLMs inference.

To address this challenge, numerous KV cache compression techniques have been explored, among which low-rank methods — particularly Singular Value Decomposition (SVD) — stand out for their effectiveness (Li et al., 2025). These approaches are motivated by the empirical observation that the KV cache often exhibits strong low-rank structures (Yu et al., 2024), which can be exploited to reduce memory usage. However, existing low-rank methods face a trade-off between efficiency and adaptivity. Fixed post-training methods either derive projection bases from calibration activations or factorize KV projection weights before inference, and then reuse the resulting low-dimensional representations during serving (Chang et al., 2025; Saxena et al., 2024; Zhang et al., 2024a; Yan et al., 2025). They avoid per-prompt decomposition, but their accuracy can depend on calibration-data match and rank allocation, particularly under aggressive compression (Sun et al., 2025; Wang et al., 2025b; Zhu et al., 2026). Prompt-dependent methods estimate or adapt low-rank representations during inference, which improves robustness to the current context but introduces additional computation, especially during prefilling (Sun et al., 2025; Chang et al., 2026; Zhu et al., 2026).

Our key insight is that the dominant KV subspace of a long prompt need not be estimated from either an external calibration set or the entire prompt. Instead, it can be approximated from a carefully selected subset of prompt tokens. At the same time, KV cache entries should not be compressed or reconstructed uniformly. Sink tokens

*Corresponding author.

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 2997–3017
November 6-10, 2026 ©2026 Association for Computational Linguistics

serve as stable global pivots in attention and should be preserved in full precision (Xiao et al., 2024), while most non-sink tokens can be stored in a low-rank coefficient space and reconstructed only when they are likely to matter for the current decoding step.

Based on this insight, we propose **S^{4}R**, a prompt-aware low-rank KV cache compression method with selective sampling and sparse reconstruction. During prefilling, S^{4}R keeps sink tokens in full precision, samples representative non-sink tokens from the prompt, normalizes their KV states, and constructs key/value subspaces with truncated SVD. It then stores non-sink keys and values as low-rank coefficients rather than full-dimensional KV states. During decoding, S^{4}R estimates token relevance directly in the latent coefficient space, combines a local window with globally important positions, and reconstructs only the selected KV entries for final attention computation. This design combines the low prefilling overhead of fixed compression with the context adaptivity of prompt-dependent methods.

We evaluate S^{4}R on LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024) with Llama (Grattafiori et al., 2024) and Qwen (Yang et al., 2025b; Qwen et al., 2025; Yang et al., 2025a) model families. On LongBench, S^{4}R achieves up to **5**$\times$ KV cache compression and remains close to the full-cache baseline across model scales. Across accuracy and efficiency evaluations, S^{4}R consistently matches or outperforms prior compressed-cache baselines while substantially reducing long-context inference latency.

- We propose S^{4}R, a prompt-aware low-rank KV cache compression method that builds compact key/value subspaces from selectively sampled prompt tokens and combines sink-token preservation, latent-space relevance estimation, and sparse reconstruction to reduce persistent KV memory while preserving context adaptivity.
- We evaluate S^{4}R on LongBench and RULER, showing near full-cache accuracy under up to **5**$\times$ KV compression and a favorable accuracy–efficiency trade-off against strong compressed-cache baselines.

2 Related Work

Low-rank KV cache compression reduces redundancy along the hidden or channel dimension, and is complementary to methods that shorten the cached sequence, quantize cache entries, merge tokens, or move them across the memory hierarchy (Li et al., 2025). The most relevant low-rank methods differ along three axes: where the compression subspace is obtained, how rank is allocated across keys, values, heads, or layers, and whether decoding requires reconstructing the full cache.

Pre-computed subspaces and adaptive rank allocation. Several methods determine most of the low-rank representation before serving the current prompt. Palu and LoRC factorize key/value projection weights and cache compact intermediate states, with LoRC using progressive layer-wise compression to control accumulated error (Chang et al., 2025; Zhang et al., 2024a). Eigen Attention derives bases from calibration activations and performs attention in the induced low-dimensional space, while ReCalKV treats keys and values separately through head reordering, grouped SVD, offline calibration, and matrix fusion (Saxena et al., 2024; Yan et al., 2025). CSKV adds lightweight reconstruction training and a bi-branch cache that preserves a recent full-precision window (Wang et al., 2024b). KQ-SVD instead chooses the projection by directly minimizing query–key inner-product distortion and gives a closed-form low-rank solution for preserving pre-softmax attention scores (Lesens et al., 2026). Other work adapts the compression granularity: TALE assigns token-dependent ranks and avoids eager reconstruction (Lee et al., 2025), STAR-KV learns fine-grained ranks through differentiable singular-value thresholding (Bhatnagar et al., 2026), and CommonKV uses SVD-based parameter sharing with adaptive cross-layer budget allocation (Wang et al., 2025b). These approaches reduce or remove prompt-time decomposition, but their bases, rank policies, or calibration statistics are not constructed solely from the current prompt.

Prompt-adaptive subspaces. Prompt-dependent methods align the compressed representation with the current context, at the cost of online decomposition or updates. ShadowKV applies SVD to current-prompt pre-RoPE keys, stores a low-rank key cache on GPU, offloads values, and reconstructs selected sparse KV pairs during decoding (Sun et al., 2025). xKV performs prompt-time cross-layer SVD and shares aligned low-rank structure across grouped layers (Chang et al., 2026). OjaKV continuously adapts its basis with online PCA, using a comprehensive prefilling update, lightweight periodic de-

Algorithm 1 S^{4}R Prefilling Stage

Input: Non-sink pre-RoPE states $\mathbf{S}^K, \mathbf{S}^V$; recent query window $\widetilde{\mathbf{Q}}$; sampling budget T', and rank r; $H \in \{K, V\}$

Output: Low-rank bases $\mathbf{B}^K, \mathbf{B}^V$, coefficient caches $\mathbf{C}^K, \mathbf{C}^V$, and query coefficient window $\mathbf{C}^Q$

$\mathcal{I} \leftarrow \textsc{HybridSample}(T, T')$
$\mathbf{S}^H_{[\mathcal{I}]} \leftarrow \textsc{Normalize}(\mathbf{S}^H_{[\mathcal{I}]}, \text{dim} = -1)$
$(\mathbf{U}^H, \mathbf{\Sigma}^H, \mathbf{B}^H) \leftarrow \text{SVD}_r\left(\mathbf{S}^H_{[\mathcal{I}]}\right)$
$\mathbf{C}^H \leftarrow \mathbf{S}^H \mathbf{B}^H$ ▷ Project all non-sink states
$\mathbf{C}^Q \leftarrow \widetilde{\mathbf{Q}} \mathbf{B}^K$ ▷ Initialize the latent query window
return $\mathbf{B}^K, \mathbf{B}^V, \mathbf{C}^K, \mathbf{C}^V, \mathbf{C}^Q$

coding updates, and full-rank anchors (Zhu et al., 2026). S^{4}R follows this context-adaptive direction, but constructs both key and value bases from a sampled subset of non-sink prompt tokens rather than decomposing or updating over the full prompt.

Latent-space selection and sparse reconstruction. Low-rank representations can also serve as inexpensive proxies for deciding which tokens merit full attention. Loki uses compressed keys to approximate attention scores for sparse token selection, but does not itself reduce the persistent value cache (Singhania et al., 2024). SALS stores a compressed latent cache, scores tokens through RoPE-free query–key interactions in that space, and reconstructs only a selected subset (Mu et al., 2025). S^{4}R shares this reconstruction-aware design, while coupling it with prompt-sampled key/value bases, full-precision sink tokens, a local query window, and a persistent coefficient cache for both keys and values. It therefore occupies the intersection of prompt-adaptive low-rank compression and latent-space sparse reconstruction. Appendix C covers complementary eviction, merging, quantization, and system-oriented methods.

3 Method

S^{4}R consists of two main stages: the *prefilling* and the *decoding* stages. We defer background on standard KV cache computation and SVD to Appendix A.

3.1 Prefilling Stage

The main motivation of the prefilling stage is to efficiently construct suitable low-rank subspaces for keys and values. As directly applying SVD to the full long-context sequence (e.g., 128k tokens) is computationally prohibitive, this stage starts by selecting a representative subset of tokens from the input prompt to approximate the dominant subspace structure of the full KV states. Together with the full-precision preservation of sink tokens, Algorithm 1 summarizes the prompt-aware subspace construction and coefficient-cache initialization performed during prefilling.

3.1.1 Sink Token Handling

Due to their role as global pivots in attention, sink tokens typically exhibit consistently high attention weights across queries and layers (Xiao et al., 2024). As a result, their representations are critical for maintaining stable long-range dependencies and cannot be reliably approximated by a low-rank subspace without incurring noticeable information loss.

To preserve their fidelity, we exclude the first s sink tokens from the compression pipeline and store their pre-RoPE key and value representations in full precision. The remaining tokens are treated as non-sink tokens and are subject to low-rank approximation.

3.1.2 Subspace Construction

Let the non-sink tokens in the input prompt be denoted as $\mathbf{X} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}\times\mathcal{D}}$. We construct a subset of tokens $\mathbf{X}_{[\mathcal{I}]} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}'\times\mathcal{D}}$ by selecting $\mathcal{T}' \ll \mathcal{T}$ using a hybrid sampling strategy that combines (i) uniform sampling over early positions and (ii) a contiguous block of most recent (right) tokens. Here, $\mathcal{I} \in \mathbb{N}^{\mathcal{B}\times\mathcal{T}'}$ denotes the indices of the selected non-sink tokens. Correspondingly, the non-sink tokens are partitioned into two disjoint subsets: the selected tokens ($\mathbf{X}_{[\mathcal{I}]}$) and the remaining tokens ($\mathbf{X}_{[\widetilde{\mathcal{I}}]}$), where $\widetilde{\mathcal{I}}$ denotes the complement of $\mathcal{I}$ along the sequence dimension.

Before applying SVD to the selected tokens, we perform token-wise normalization to mitigate the influence of varying vector magnitudes on the learned subspace. In practice, tokens with larger norms may dominate the decomposition and bias the resulting singular vectors toward high-energy directions. To alleviate this effect, we normalize each sampled pre-RoPE key and value representation along the feature dimension:

$$\mathbf{S}^{K/V}_{[\mathcal{I}]} \leftarrow \textsc{normalize}(\mathbf{S}^{K/V}_{[\mathcal{I}]}, \text{dim}= -1). \quad (1)$$

This normalization step ensures that the subspace primarily captures directional information rather than being skewed by magnitude differences, leading to a more stable and representative low-rank approximation.

Then, we perform truncated SVD for each layer:

$$\mathbf{U}^{K/V}, \mathbf{\Sigma}^{K/V}, \mathbf{B}^{K/V} \leftarrow \mathrm{SVD}_r\left(\mathbf{S}^{K/V}_{[\mathcal{I}]}\right), \quad (2)$$

where $\mathbf{U}^{K/V} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}'\times r}$ are the left singular vectors, $\mathbf{\Sigma}^{K/V} \in \mathbb{R}^{\mathcal{B}\times r\times r}$ are the singular values, and $\mathbf{B}^{K/V} \in \mathbb{R}^{\mathcal{B}\times\mathcal{D}'\times r}$ are the top-r right singular vectors, which serve as the low-rank bases.

We then project the non-sink key and value representations onto the learned subspaces to obtain their coefficient representations:

$$\mathbf{C}^{K/V} = \mathbf{S}^{K/V}\mathbf{B}^{K/V}, \quad (3)$$

where $\mathbf{C}^{K/V} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}\times r}$ are the projection coefficients.

3.1.3 Initial Local Query Window

To enable efficient relevance estimation without reconstructing the full keys and values during decoding, we also project the local queries into the same low-rank subspace.

Let the most recent queries be denoted as $\widetilde{\mathbf{Q}} \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{W}\times\mathcal{D}'}$. The projected query coefficients are then computed as

$$\mathbf{C}^{Q} = \widetilde{\mathbf{Q}}\mathbf{B}^{K} \quad (4)$$

where $\mathbf{C}^{Q} \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{W}\times r}$.

3.2 Decoding Stage

At a high level, decoding in S^{4}R keeps the KV cache compressed while reconstructing only a small set of useful tokens at each step. Each new token is projected into the prefilling-stage low-rank subspace to update the coefficient cache. Recent query coefficients are then used to score cached tokens in latent space, which provides an inexpensive estimate of token relevance. S^{4}R reconstructs only the selected local and globally important non-sink tokens, concatenates them with the full-precision sink tokens, and performs attention over this compact set. Algorithm 2 summarizes the per-step S^{4}R decoding procedure, including latent-space relevance estimation, token selection, and sparse KV reconstruction.

Algorithm 2 S^{4}R Decoding at Step t

Input: New pre-RoPE states $\mathbf{s}_t^Q, \mathbf{s}_t^K, \mathbf{s}_t^V$; bases $\mathbf{B}^K, \mathbf{B}^V$; coefficient caches $\mathbf{C}_{t-1}^K, \mathbf{C}_{t-1}^V$; query window $\mathbf{C}^Q$; sink states $\mathbf{K}_{\text{sink}}, \mathbf{V}_{\text{sink}}$; W, r, κ, and ρ; $H \in \{K, V\}$

Output: Attention output $\mathbf{O}_t$ and updated $\mathbf{C}_t^K, \mathbf{C}_t^V, \mathbf{C}^Q$

$\mathbf{c}_t^{Q/K} \leftarrow \mathbf{s}_t^{Q/K}\mathbf{B}^K$

$\mathbf{c}_t^V \leftarrow \mathbf{s}_t^V\mathbf{B}^V$ ▷ Project the new states

$\mathbf{C}_t^H \leftarrow \mathbf{C}_{t-1}^H \oplus \mathbf{c}_t^H$ ▷ Append to the coefficient cache

$\mathbf{C}^Q \leftarrow \textsc{concat}(\mathbf{C}^Q[:, :, 1:, :], \mathbf{c}_t^Q)$ ▷ Update the FIFO query window

$L_t \leftarrow T + t$

$\mathbf{A} \leftarrow \sigma_{L_t}\left(\mathrm{Mask}_{\text{causal}}(\frac{\mathbf{C}^Q\left(\mathbf{C}_t^K\right)^\top}{\sqrt{r}})\right)$

$\widetilde{\mathbf{A}} \leftarrow \mathcal{F}_P\left(\frac{1}{W}\sum_{i=1}^{W}\mathbf{A}[:, :, i, :]; \kappa\right)$ ▷ Pooling

$\mathcal{I}_{\text{local}} \leftarrow \{L_t - W, \ldots, L_t - 1\}$

$k \leftarrow \max\left(\lfloor \rho L_t \rfloor - W, 0\right)$

$\mathcal{I}_{\text{global}} \leftarrow \textsc{TopK}(\widetilde{\mathbf{A}}[:, : L_t - W], k)$

$\widehat{\mathcal{I}} \leftarrow \mathcal{I}_{\text{local}} \cup \mathcal{I}_{\text{global}}$

$\widehat{\mathbf{H}} \leftarrow \mathbf{C}^H_{t\,[\widehat{\mathcal{I}}]}\left(\mathbf{B}^H\right)^\top$ ▷ Reconstruct only selected positions

$\mathbf{H}^\star \leftarrow \textsc{concat}(\mathbf{H}_{\text{sink}}, \widehat{\mathbf{H}})$

$\mathbf{O}_t \leftarrow \sigma\left(\frac{\mathrm{RoPE}(\mathbf{s}_t^Q)\,\mathrm{RoPE}(\mathbf{K}^\star)^\top}{\sqrt{d_h}}\right)\mathbf{V}^\star$ ▷ Use the original token positions

return $\mathbf{O}_t, \mathbf{C}_t^K, \mathbf{C}_t^V, \mathbf{C}^Q$

3.2.1 Compressed Cache Update

Let $\mathbf{s}_t^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times 1\times\mathcal{D}'}$ and $\mathbf{s}_t^{K/V} \in \mathbb{R}^{\mathcal{B}\times 1\times\mathcal{D}'}$ denote the query, key, and value representations at step t, respectively. The corresponding coefficient representations are computed as:

$$\begin{aligned}\mathbf{c}_t^{Q/K} &= \mathbf{s}_t^{Q/K}\mathbf{B}^K,\\ \mathbf{c}_t^{V} &= \mathbf{s}_t^{V}\mathbf{B}^V,\end{aligned} \quad (5)$$

where $\mathbf{c}_t^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times 1\times r}$ and $\mathbf{c}_t^{K/V} \in \mathbb{R}^{\mathcal{B}\times 1\times r}$.

The key and value coefficients are appended to the cached sequences:

$$\mathbf{C}_t^{K/V} = \mathbf{C}_{t-1}^{K/V} \oplus \mathbf{c}_t^{K/V}, \quad (6)$$

where $\mathbf{C}_t^{K/V} \in \mathbb{R}^{\mathcal{B}\times(\mathcal{T}+t)\times r}$ stores the accumulated coefficients.

In contrast, query coefficients are maintained as a fixed-size sliding window of length $\mathcal{W}$, forming a

first-in-first-out queue $\mathbf{C}^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{W}\times r}$. At each step t, the new coefficient $\mathbf{c}_t^Q$ is appended, while the oldest entry is removed:

$$\mathbf{C}^Q \leftarrow \textsc{concat}(\mathbf{C}^Q[:,:,1:,:], ; \mathbf{c}_t^Q). \quad (7)$$

3.2.2 Latent-Space Relevance Estimation

To avoid explicit reconstruction of the full key and value representations, we approximate the attention scores directly in the latent coefficient space.

Let $\mathcal{L}_t = \mathcal{T} + t$ denote the current non-sink cache length. Given the query coefficients $\mathbf{C}^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{W}\times r}$ and the cached key coefficients $\mathbf{C}_t^K \in \mathbb{R}^{\mathcal{B}\times\mathcal{L}_t\times r}$, we first compute the latent-space attention logits:

$$\mathbf{Z} = \frac{\mathbf{C}^Q(\mathbf{C}_t^K)^\top}{\sqrt{r}}, \quad (8)$$

where the inner product is taken along the latent dimension r, yielding $\mathbf{Z} \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{W}\times\mathcal{L}_t}$.

We then apply the causal mask before normalization, so each query only attends to valid historical positions:

$$\mathcal{A} = \sigma_{\mathcal{L}_t}\left(\mathrm{Mask}_{\text{causal}}(\mathbf{Z})\right), \quad (9)$$

where the softmax (σ) is computed over the cache-length dimension.

To obtain a single relevance score for each cached token, we average the attention scores over the query window:

$$\mathcal{A}^Q = \frac{1}{\mathcal{W}}\sum_{i=1}^{\mathcal{W}} \mathcal{A}_{[:,:,i,:]}. \quad (10)$$

Here, $\mathcal{A}^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{L}_t}$.

3.2.3 Sparse Reconstruction and Attention

Given the relevance scores $\mathcal{A}^Q \in \mathbb{R}^{\mathcal{B}\times\mathcal{G}\times\mathcal{L}_t}$, we first apply a pooling operation along the sequence dimension before selecting global tokens. This follows SnapKV-style token clustering (Li et al., 2024), where neighboring tokens are smoothed or grouped to produce more stable importance estimates:

$$\widetilde{\mathcal{A}} = \mathcal{F}_{\mathcal{P}}(\mathcal{A}^Q; \kappa) \quad (11)$$

where $\mathcal{F}_{\mathcal{P}}$ denotes either average pooling or max pooling with kernel size κ.

For models with $\mathcal{G} > 1$, we further aggregate across the group dimension using $\mathcal{F}_{\mathcal{G}} \in \{\mathrm{mean}, \max\}$:

$$\widetilde{\mathcal{A}} = \mathcal{F}_{\mathcal{G}}(\widetilde{\mathcal{A}}), \quad (12)$$

where $\widetilde{\mathcal{A}} \in \mathbb{R}^{\mathcal{B}\times\mathcal{L}_t}$ represents the final relevance score for each cached token.

We select a subset of historical tokens for reconstruction by combining a local window with globally important tokens. We always retain the most recent $\mathcal{W}$ tokens:

$$\mathcal{I}_{\text{local}} = \{\mathcal{L}_t - \mathcal{W}, \ldots, \mathcal{L}_t - 1\}. \quad (13)$$

From the remaining tokens, we select the top-k positions according to $\widetilde{\mathcal{A}}$:

$$\mathcal{I}_{\text{global}} = \textsc{TopK}(\widetilde{\mathcal{A}}_{[:,:\mathcal{L}_t-\mathcal{W}]}, k), \quad (14)$$

where $k = \max(\lfloor \rho\mathcal{L}_t \rfloor - \mathcal{W}, 0)$ is determined by the reconstruction ratio ρ.

The final index set is given by:

$$\widehat{\mathcal{I}} = \mathcal{I}_{\text{local}} \cup \mathcal{I}_{\text{global}}. \quad (15)$$

Given the selected indices, we reconstruct the corresponding key and value representations from the low-rank coefficients:

$$\widehat{\mathbf{K}} = \mathbf{C}_{t\,[\widehat{\mathcal{I}}]}^K(\mathbf{B}^K)^\top, \quad \widehat{\mathbf{V}} = \mathbf{C}_{t\,[\widehat{\mathcal{I}}]}^V(\mathbf{B}^V)^\top, \quad (16)$$

where $\widehat{\mathbf{K}}, \widehat{\mathbf{V}} \in \mathbb{R}^{\mathcal{B}\times|\widehat{\mathcal{I}}|\times\mathcal{D}'}$.

These reconstructed tokens are then concatenated with the sink tokens:

$$\begin{aligned}\mathbf{K}^\star &= \textsc{concat}(\mathbf{K}_{\text{sink}}, \widehat{\mathbf{K}}),\\ \mathbf{V}^\star &= \textsc{concat}(\mathbf{V}_{\text{sink}}, \widehat{\mathbf{V}}).\end{aligned} \quad (17)$$

We then apply RoPE to $\mathbf{K}^\star$ according to their original positions to ensure positional consistency.

The final attention output is computed as:

$$\mathbf{O}_t = \sigma\left(\frac{\mathrm{RoPE}(\mathbf{s}_t^Q)\mathrm{RoPE}(\mathbf{K}^\star)^\top}{\sqrt{d_h}}\right)\mathbf{V}^\star. \quad (18)$$

3.3 Memory Analysis

We analyze the per-layer memory footprint in number of stored scalar elements, omitting the batch dimension for clarity. At decoding step t, the full KV cache stores both keys and values for the s sink tokens and the $\mathcal{L}_t$ non-sink tokens:

$$M_{\text{full}}(t) = 2(s + \mathcal{L}_t)\mathcal{D}'. \quad (19)$$

In S^{4}R, sink tokens are kept in full precision, while non-sink tokens are stored in the low-rank coefficient cache. The persistent cache therefore consists of the full sink KV states, the key/value

coefficient caches, the query-coefficient window, and the learned key/value bases:

$$M_{\mathrm{S^4R}}(t) = 2s\mathcal{D}' + 2\mathcal{L}_t r + \mathcal{G}\mathcal{W}r + 2\mathcal{D}'r. \quad (20)$$

The term $\mathcal{G}\mathcal{W}r$ stores the fixed-size query-coefficient window, while the last term stores $\mathbf{B}^K$ and $\mathbf{B}^V$. The resulting persistent memory ratio is

$$\frac{M_{\mathrm{S^4R}}(t)}{M_{\mathrm{full}}(t)} = \frac{s\mathcal{D}' + \mathcal{L}_t r + \frac{1}{2}\mathcal{G}\mathcal{W}r + \mathcal{D}'r}{(s + \mathcal{L}_t)\mathcal{D}'}. \quad (21)$$

When $\mathcal{L}_t \gg s, r$, the ratio approaches $r/\mathcal{D}'$, showing that the long-context memory is dominated by the low-rank coefficient dimension.

During attention computation, S^{4}R reconstructs only the selected non-sink tokens. Let

$$\begin{aligned} m_t = |\widehat{\mathcal{I}}| &= \mathcal{W} + \max(\lfloor \rho\mathcal{L}_t \rfloor - \mathcal{W}, 0) \\ &= \max(\mathcal{W}, \lfloor \rho\mathcal{L}_t \rfloor) \end{aligned} \quad (22)$$

denote the number of reconstructed non-sink tokens. The transient working memory is then

$$M_{\mathrm{work}}(t) = 2m_t\mathcal{D}', \quad (23)$$

which corresponds to the reconstructed key/value buffer. Thus, the peak per-step memory is approximately

$$M_{\mathrm{peak}}(t) \approx M_{\mathrm{S^4R}}(t) + M_{\mathrm{work}}(t). \quad (24)$$

Since $m_t \ll \mathcal{L}_t$ and $r \ll \mathcal{D}'$, S^{4}R reduces the persistent KV cache substantially while avoiding full-cache reconstruction during decoding.

4 Experiments

In this section, we comprehensively evaluate the effectiveness and memory savings of S^{4}R across multiple benchmarks. We also conduct speed evaluations to analyze the impact of our method on generation efficiency. Additionally, we perform ablation studies to analyze the contributions of key components within the S^{4}R method.

Unless otherwise specified, the rank r is obtained by solving Eq. (21) for the target retention ratio η, and the reconstruction ratio is set to $\rho = \eta$. We use $s = 4$ sink tokens following the default setting of Xiao et al. (2024), a query-window size of $\mathcal{W} = 32$, and a pooling kernel size of $\kappa = 7$ following the default setting of Li et al. (2024). For token relevance aggregation, we use max pooling for $\mathcal{F}_{\mathcal{P}}$ and mean aggregation for $\mathcal{F}_{\mathcal{G}}$. The number of sampled tokens $\mathcal{T}'$ is determined by the product of the total prompt length and the target retention ratio, with a 7:3 recent-to-uniform sampling split.

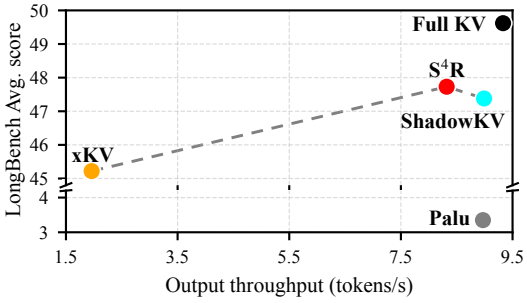


Figure 1: Performance–speed trade-off on Qwen3-4B. The y-axis is broken to include the dominated Palu.

4.1 LongBench and RULER Results

LongBench Figure 1 summarizes the main accuracy–efficiency trade-off on Qwen3-4B by combining the LongBench average scores in Table 1 with the output-throughput measurements in Table 3. S^{4}R lies on the favorable part of the trade-off curve: it achieves a LongBench average score of 47.73 (Table 7) under the latency-oriented efficiency setting, close to the default LongBench result of 47.79 in Table 1, while reaching 8.32 output tokens/s in Table 3. In contrast, xKV has a lower Qwen3-4B LongBench average score in Table 1 and is much slower in Table 3, with output throughput dropping to 1.96 tokens/s and 512-token latency increasing to 260.76s. Palu and ShadowKV are faster, but Table 1 shows that Palu collapses under aggressive compression and ShadowKV remains below S^{4}R on Qwen3-4B accuracy.

Table 1 further expands this comparison across model families. S^{4}R substantially outperforms Palu across all evaluated models, showing that prompt-aware subspace construction is important when the KV cache is compressed to 20–30% of its original size. Compared with ShadowKV and xKV, S^{4}R achieves the best average score on Llama-3.2-1B and Qwen3-4B, and remains competitive on larger models. On Llama-3.1-8B and Qwen2.5-14B, xKV obtains slightly higher average scores, but this accuracy difference should be interpreted together with the efficiency gap. Overall, Figure 1, Table 1, and Table 3 show that S^{4}R preserves most of the LongBench accuracy of stronger prompt-dependent low-rank baselines while avoiding their large runtime overhead.

RULER Table 2 reports the RULER results under a 2.5× compression ratio and a 128k context window. In addition to low-rank baselines, we include two representative eviction-based methods, StreamingLLM (SLM) and SnapKV, to compare

Methods	η	Fewshot			Single-Doc QA			Multi-Doc QA			LCC	PRE	Avg.
		SSM	Trec	TQA	MQA	NQA	QPR	2Wiki	HQA	MSQ			
Llama-3.2-1B	1	28.73	63.50	80.35	41.09	18.29	13.02	29.74	31.73	14.65	23.76	2.50	31.58
Palu		0.49	0.50	2.12	3.08	0.85	1.71	1.54	1.23	0.96	13.05	0.83	2.40
ShadowKV		19.97	45.00	73.55	30.67	14.96	11.62	23.33	29.08	9.79	19.46	5.50	25.72
xKV-2	0.2	17.14	32.50	75.20	35.78	11.82	11.74	25.92	30.00	11.05	15.81	5.04	24.73
xKV-4		20.58	35.50	77.08	36.46	14.87	11.81	25.78	28.92	12.74	16.33	6.01	26.01
S^4R		25.91	53.00	74.97	37.14	13.29	11.63	25.83	29.02	11.34	23.43	3.39	28.09
Qwen3-4B	1	35.91	67.50	87.72	52.17	27.88	45.51	42.76	57.44	23.75	5.18	100.00	49.62
Palu		0.89	2.50	4.43	3.83	0.60	1.64	0.92	0.75	0.83	19.82	0.64	3.35
ShadowKV		34.14	68.50	85.08	48.08	25.80	42.79	39.60	57.67	24.35	5.12	90.00	47.38
xKV-2	0.2	32.88	41.00	88.06	51.02	22.80	41.58	38.95	53.80	19.21	4.92	89.75	44.00
xKV-4		34.27	36.50	86.54	50.58	25.69	42.78	42.28	54.91	22.41	4.95	96.50	45.22
S^4R		34.38	72.50	89.31	50.92	23.81	40.85	39.93	53.90	18.84	5.05	96.17	47.79
Llama-3.1-8B	1	43.96	73.50	92.29	56.34	30.51	45.59	50.02	58.22	31.75	24.76	100.00	55.18
Palu		4.99	0.00	2.58	2.23	0.84	3.00	1.45	0.98	0.72	24.51	0.27	3.78
ShadowKV		45.10	68.00	92.34	35.65	29.84	24.81	49.24	58.13	31.02	25.08	92.00	50.11
xKV-2	0.2	34.52	65.00	89.92	56.60	31.27	44.06	48.52	57.68	29.96	32.10	99.50	53.56
xKV-4		35.16	63.50	91.00	56.94	32.27	45.96	50.21	57.99	30.61	32.51	100.00	54.19
S^4R		33.42	59.50	92.83	54.46	30.81	41.68	49.47	57.92	31.44	32.29	99.50	53.03
Qwen2.5-14B	1	38.48	78.50	90.12	52.93	26.79	44.94	58.99	61.14	38.54	20.17	98.50	55.37
Palu		20.76	0.64	15.29	22.32	3.42	8.44	10.63	9.23	5.51	16.59	28.33	12.83
ShadowKV		36.52	69.25	80.05	45.96	27.50	40.74	56.31	60.67	38.86	18.85	98.50	52.11
xKV-2	0.3	37.17	77.50	91.02	51.50	27.08	45.06	57.02	59.50	38.51	19.41	99.00	54.80
xKV-4		38.10	76.00	90.18	50.97	27.07	43.95	57.61	61.13	39.12	19.52	97.92	54.69
S^4R		38.03	72.50	89.61	48.02	27.08	39.49	56.33	59.78	39.52	20.25	94.50	53.19

Table 1: Performance comparison of different models and methods on LongBench.

against token-retention strategies. S^4R achieves an average score of 93.61, nearly matching the full KV cache result of 93.97 and slightly outperforming xKV at 93.47. It also improves over Palu, ShadowKV, SnapKV, and SLM by 11.39, 4.00, 4.80, and 45.22 points on average, respectively. The gains are especially clear on multi-key and multi-query retrieval tasks, where S^4R preserves perfect accuracy, suggesting that the sampled low-rank subspace and sparse reconstruction strategy retain the information needed for long-context retrieval. The main remaining gap appears on FWE, where S^4R trails full KV and xKV, indicating that some tasks remain sensitive to the exact retained or reconstructed token set.

Method	MK1	MK2	MQ	MV	VT	FWE	Avg.
Full KV	100.00	100.00	100.00	97.50	83.00	83.33	93.97
SLM	55.00	30.00	35.00	35.00	47.00	88.33	48.39
SnapKV	100.00	70.00	100.00	97.50	87.00	78.33	88.81
Palu	95.00	85.00	92.50	97.50	70.00	53.33	82.22
ShadowKV	100.00	90.00	97.50	92.50	91.00	66.67	89.61
xKV	100.00	100.00	97.50	100.00	80.00	83.33	93.47
S^4R	100.00	100.00	100.00	100.00	85.00	76.67	93.61

Table 2: Comparison of different methods on RULER using Qwen2.5-7B-Instruct-1M under a 2.5× compression ratio and a 128k context window.

4.2 Efficiency Evaluation

We report model-level generation efficiency using TTFT, average latency for generating 512 output tokens, time per output token (TPOT), output throughput, and total throughput.

Table 3 shows that S^4R introduces moderate overhead compared with the full KV baseline and fixed/offline-style methods, mainly because it performs latent-space scoring and sparse reconstruction during decoding. Nevertheless, the overhead is small in this setting: S^4R reaches a 512-token latency of 61.54s, close to Full KV, Palu, and ShadowKV at 54.89s, 57.06s, and 56.92s, respectively. More importantly, S^4R is substantially more efficient than xKV: it reduces TTFT from 79.90s to 27.65s and the 512-token generation latency from 260.76s to 61.54s, while improving output throughput from 1.96 to 8.32 tokens/s. This supports the intended trade-off of S^4R: compared with more expensive prompt-dependent low-rank approaches, it retains prompt adaptivity while avoiding the cost of decomposing or processing the full prompt during inference. Compared with Palu and ShadowKV, S^4R is slightly slower in this unoptimized implementation, but the LongBench and RULER results

Method	TTFT (ms) ↓	Latency$_{512}$ (s) ↓	TPOT (ms/token) ↓	Output Thr. (tokens/s) ↑	Total Thr. (tokens/s) ↑
Full KV	24737.54	54.89	59.01	9.33	2018.49
Palu	24579.62	57.06	63.57	8.97	1941.67
ShadowKV†	36143.18	56.92	40.66	8.99	1946.43
xKV	79897.02	260.76	353.95	1.96	424.89
S^{4}R‡	27650.96	61.54	66.19	8.32	1772.7

Table 3: Model-level efficiency comparison on Qwen3-4B using a 120k-token context sampled from GovReport. Latency$_{512}$ denotes the average end-to-end latency for generating 512 output tokens. All methods are evaluated without system-level, kernel-level, or operator-level optimizations. Appendix B provides the corresponding theoretical efficiency estimates.

† For ShadowKV, we keep values on GPU rather than following its standard CPU-offloading implementation, which removes CPU–GPU value-loading communication overhead and therefore gives ShadowKV a more favorable efficiency estimate.
‡ For S^{4}R, this efficiency experiment uses a latency-oriented query window $\mathcal{W} = 4$. Appendix D.1 shows that this has negligible impact on Qwen3-4B accuracy, with an average score of 47.73 compared with 47.79 for $\mathcal{W} = 32$.

show that the additional runtime is exchanged for stronger accuracy under aggressive KV compression.

4.3 Ablation Study

We ablate three components that are central to S^{4}R in the main text. Unless otherwise specified, ablation tables report grouped LongBench scores: Fewshot, SDQA, and MDQA denote the average score within each task group, while Avg. is computed over the same 11 tasks as Table 1. Additional ablations are provided in Appendix D.

4.3.1 Importance of Sink Token

S^{4}R treats the first few tokens as attention sinks and keeps their key/value states in full precision. Unlike ordinary non-sink tokens, compressing these positions into a low-rank coefficient space may introduce errors that are repeatedly exposed to many later queries. We therefore isolate the effect of sink-token preservation by comparing the default setting, which stores the first $s = 4$ sink tokens in full precision, with a variant that applies the same low-rank compression pipeline to all tokens.

Table 4 reports the grouped LongBench results. Retaining sink tokens in full precision improves the average score across all four models. The gain is largest on smaller models: the average score increases from 24.60 to 28.09 on Llama-3.2-1B and from 45.58 to 47.79 on Qwen3-4B. The improvement remains consistent on larger models, with gains of 0.71 points on Llama-3.1-8B and 1.15 points on Qwen2.5-14B. These results indicate that the first few sink tokens provide a stable global context that is difficult to recover once compressed into the same low-rank coefficient space as ordinary tokens. The task-level effects are also informative: Llama-3.2-1B benefits most on Fewshot and LCC,

Sink	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
w/	51.29	20.69	22.06	23.43	3.39	28.09
w/o	42.21	18.82	20.86	20.56	4.36	24.60
Qwen3-4B						
w/	65.40	38.53	37.56	5.05	96.17	47.79
w/o	62.99	37.00	38.11	5.04	82.00	45.58
Llama-3.1-8B						
w/	61.92	42.32	46.28	32.29	99.50	53.03
w/o	60.51	41.92	45.71	31.59	99.50	52.32
Qwen2.5-14B						
w/	66.71	38.20	51.88	20.25	94.50	53.19
w/o	64.88	37.29	50.56	19.80	94.50	52.04

Table 4: Effect of retaining sink tokens in full precision on grouped LongBench performance. "w/" keeps the first $s = 4$ sink tokens uncompressed, whereas "w/o" applies the low-rank compression pipeline to all tokens.

while Qwen3-4B shows a large improvement on passage retrieval, where PRE increases from 82.00 to 96.17. Since the sink cache only contributes $2s\mathcal{D}'$ elements per layer in Eq. (21), its memory overhead is negligible for long contexts.

4.3.2 Effect of Token-wise Normalization

We ablate the token-wise normalization step before SVD in Table 5. On Llama-3.1-8B, removing normalization leads to a slightly higher average score, with 53.32 compared to 53.03 for the default normalized variant. However, the effect is model-dependent: on Qwen2.5-14B, removing normalization causes a substantial drop from 53.19 to 46.27. The degradation is especially pronounced on single-document QA, multi-document QA, and passage retrieval, indicating that unnormalized high-magnitude KV states can dominate the SVD basis and produce less transferable low-rank subspaces. We therefore keep token-wise normalization as

Norm.	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.1-8B						
w/	61.92	42.32	46.28	32.29	99.50	53.03
w/o	62.78	42.41	46.13	32.54	100.00	53.32
Qwen2.5-14B						
w/	66.71	38.20	51.88	20.25	94.50	53.19
w/o	63.37	31.43	43.01	18.77	76.79	46.27

Table 5: Effect of token-wise normalization before SVD. Norm. indicates whether sampled KV states are normalized before constructing the subspaces.

$\mathcal{F}_{\mathcal{P}}$	$\mathcal{F}_{\mathcal{G}}$	Fewshot	SDQA	MDQA	Group Avg.
Llama-3.2-1B					
max	mean	51.29	20.69	22.06	31.35
max	max	51.27	19.77	22.67	31.24
avg	mean	52.24	20.03	22.52	31.60
avg	max	51.92	20.25	22.69	31.62
Qwen3-4B					
max	mean	65.40	38.53	37.56	47.16
max	max	65.36	37.89	37.56	46.94
avg	mean	64.76	38.30	37.74	46.93
avg	max	65.31	38.59	37.38	47.09
Llama-3.1-8B					
max	mean	61.92	42.32	46.28	50.17
max	max	61.70	42.47	46.66	50.28
avg	mean	60.69	42.49	46.37	49.85
avg	max	60.17	42.20	46.47	49.61
Qwen2.5-14B					
max	mean	66.71	38.20	51.88	52.26
max	max	66.66	37.98	52.02	52.22
avg	mean	66.77	37.50	51.98	52.08
avg	max	66.78	37.79	52.01	52.19

Table 6: Effect of pooling and aggregation choices for latent-space relevance estimation.

the default because it improves robustness across model families, even though it is not uniformly optimal on every model.

4.3.3 Pooling and Aggregation

We study how the pooling function $\mathcal{F}_{\mathcal{P}}$ over neighboring token scores and the aggregation function $\mathcal{F}_{\mathcal{G}}$ affect the final sparse reconstruction set. Table 6 compares max and average pooling along the sequence dimension, combined with mean or max aggregation across group dimensions. Across all four model settings, the average score varies only mildly across these choices: the largest gap among the four variants is 0.38 for Llama-3.2-1B, 0.23 for Qwen3-4B, 0.67 for Llama-3.1-8B, and 0.18 for Qwen2.5-14B. This suggests that S^{4}R is not highly sensitive to the exact smoothing and group-aggregation rule used for latent-space relevance scores.

5 Conclusion

We presented S^{4}R, a prompt-aware low-rank KV cache compression method that combines selective subspace construction with sparse reconstruction. Instead of relying on external calibration data or decomposing the full prompt, S^{4}R constructs key/value bases from sampled non-sink prompt tokens, stores non-sink KV states as latent coefficients, preserves sink tokens in full precision, and reconstructs only the local window plus globally important positions during decoding. This design targets the main trade-off in low-rank KV compression: fixed methods are efficient but less adaptive, while prompt-dependent methods are adaptive but can be costly on long prompts.

Experiments on LongBench and RULER show that S^{4}R maintains strong long-context accuracy under aggressive KV compression. On LongBench, it outperforms fixed low-rank baselines and remains competitive with more expensive prompt-dependent methods; on RULER, it achieves 93.61 average score under 2.5$\times$ compression, nearly matching full KV and slightly surpassing xKV. Efficiency results further show that S^{4}R substantially reduces the overhead of xKV in a 120k-token setting, lowering TTFT from 79.90s to 27.65s and 512-token latency from 260.76s to 61.54s. Together, these results indicate that sampled-token SVD and latent-space sparse reconstruction provide a practical path toward adaptive KV cache compression for long-context LLM inference.

Limitations

Our implementation lacks system- or kernel-level optimization, so reported efficiency reflects algorithmic overhead rather than a fully optimized deployment. Experiments are limited to Llama and Qwen on LongBench and RULER, which emphasize long-input understanding with relatively short outputs; behavior under very long generation, other architectures, and broader workloads remains to be validated. S^{4}R also depends on hyperparameters and a fixed token-sampling heuristic, and tasks with scattered evidence (e.g., FWE) can still degrade when latent scoring misses relevant positions.

Acknowledgments

This work was supported by the HPC platform of ShanghaiTech University, and the Core Facility Platform of Computer Science and Communication, SIST, ShanghaiTech University.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.

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

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025. LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, Vienna, Austria. Association for Computational Linguistics.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. 2025. RocketKV: Accelerating Long-Context LLM Inference via Two-Stage KV Cache Compression. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 3358–3392, Vancouver, Canada. PMLR.

Priyansh Bhatnagar, Ashkan Moradifirouzabadi, SeHyun Yang, SeungJae Lee, Jungwook Choi, and Mingu Kang. 2026. STAR-KV: Low-Rank KV Cache Compression via Soft Thresholding for Adaptive Rank Control. In *Forty-third International Conference on Machine Learning*, Seoul, South Korea.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. In *Second Conference on Language Modeling*, Montreal, Canada. OpenReview.net.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed Abdelfattah, and Kai-Chiang Wu. 2025. Palu: KV-Cache Compression with Low-Rank Projection. In *International Conference on Learning Representations*, volume 2025, pages 50222–50249, Singapore.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. 2026. xKV: Cross-Layer KV-Cache Compression via Aligned Singular Vector Extraction. In *Forty-third International Conference on Machine Learning*, Seoul, South Korea.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024. NACL: A General and Effective KV Cache Eviction Framework for LLM at Inference Time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926, Bangkok, Thailand. Association for Computational Linguistics.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 113152–113188. Curran Associates, Inc.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The Language Model Evaluation Harness.

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

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*, volume 37, pages 1270–1303, Vancouver, Canada. Curran Associates, Inc.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? In *First Conference on Language Modeling*, Philadelphia, Pennsylvania, United States. OpenReview.net.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. 2025. SpeCache: Speculative Key-Value Caching for Efficient Generation of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 27917–27928, Vancouver, Canada. PMLR.

Vinay Joshi, Pratik Prabhanjan Brahma, Zicheng Liu, and Emad Barsoum. 2025. TaDA: Training-free recipe for Decoding with Adaptive KV Cache Compression and Mean-centering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 1435–1443, Vienna, Austria. Association for Computational Linguistics.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. GEAR: An Efficient Error Reduction Framework for KV Cache Compression in LLM Inference. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 305–321, Vancouver, Canada. PMLR.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. 2025. KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 167563–167591, San Diego, CA, USA. Curran Associates, Inc.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Jaeseong Lee, Seung-won Hwang, Aurick Qiao, Daniel Campos, Zhewei Yao, and Yuxiong He. 2025. TALE: Token-Adaptive Low-Rank KVCache Approximation with Reconstruction Elimination. *Transactions of the Association for Computational Linguistics*, 13:1298–1318.

Damien Lesens, Beheshteh T. Rakhshan, and Guillaume Rabusseau. 2026. KQ-SVD: Compressing the KV Cache with Provable Guarantees on Attention Fidelity. In *Proceedings of The 29th International Conference on Artificial Intelligence and Statistics*, volume 300 of *Proceedings of Machine Learning Research*, pages 3556–3564, Tangier, Morocco. PMLR.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, Virtual. Curran Associates, Inc.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole HU, Wei Dong, Li Qing, and Lei Chen. 2025. A Survey on Large Language Model Acceleration based on KV Cache Management. *Transactions on Machine Learning Research*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. In *Advances in Neural Information Processing Systems*, volume 37, pages 22947–22970, Vancouver, Canada. Curran Associates, Inc.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. In *Advances in Neural Information Processing Systems*, volume 37, pages 139997–140031, Vancouver, Canada. Curran Associates, Inc.

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*, volume 36, pages 52342–52364, New Orleans, LA, USA. Curran Associates, Inc.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344, Vienna, Austria. PMLR.

Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. 2025. SALS: Sparse Attention in Latent Space for KV Cache Compression. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 384–403, San Diego, CA, USA. Curran Associates, Inc.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. GPT-4 Technical Report. *Preprint*, arXiv:2303.08774.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient Context Window Extension of Large Language Models. In *International Conference on Learning Representations*, volume 2024, pages 31932–31951, Vienna, Austria.

Junjie Peng, You Wu, Haoyi Wu, Jialong Han, Xiaohua Xie, Kewei Tu, and Jianhuang Lai. 2026. GRKV:

Global Regression for Training-Free KV Cache Compression in Long-Context LLMs. *Preprint*, arXiv:2605.31105.

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, pages 606–624, Miami Beach, Florida, USA. Curan.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 Technical Report. *Preprint*, arXiv:2412.15115.

Isaac Rehg. 2024. KV-Compress: Paged KV-Cache Compression with Variable Compression Rates per Attention Head. *Preprint*, arXiv:2410.00161.

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.

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*, volume 37, pages 16692–16723, Vancouver, Canada. Curran Associates, Inc.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 57355–57373, Vancouver, Canada. PMLR.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 47901–47911, Vienna, Austria. PMLR.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, Soroosh Mariooryad, Yifan Ding, Xinyang Geng, Fred Alcober, Roy Frostig, Mark Omernick, Lexi Walker, Cosmin Paduraru, Christina Sorokin, and 1118 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Luning Wang, Shiyao Li, Xuefei Ning, Zhihang Yuan, Shengen Yan, Guohao Dai, and Yu Wang. 2024b. CSKV: Training-efficient channel shrinking for KV cache in long-context scenarios. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 468–484. PMLR.

Tuowei Wang, Minxing Huang, Fengzu Li, Ligeng Chen, Jinrui Zhang, and Ju Ren. 2025a. DynaKV: Enabling Accurate and Efficient Long-Sequence LLM Decoding on Smartphones. *Preprint*, arXiv:2511.07427.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. 2025b. CommonKV: Compressing KV Cache with Cross-layer Parameter Sharing. *Preprint*, arXiv:2508.16134.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024c. Model Tells You Where to Merge: Adaptive KV Cache Merging for LLMs on Long-Context Tasks. *Preprint*, arXiv:2407.08454.

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*, volume 2025, pages 37228–37253, Singapore.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895, Vienna, Austria.

Xianglong Yan, Zhiteng Li, Tianao Zhang, Haotong Qin, Linghe Kong, Yulun Zhang, and Xiaokang Yang. 2025. ReCalKV: Low-Rank KV Cache Compression via Head Reordering and Offline Calibration. *Preprint*, arXiv:2505.24357.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 Technical Report. *Preprint*, arXiv:2505.09388.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, and 9 others. 2025b. Qwen2.5-1M Technical Report. *Preprint*, arXiv:2501.15383.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3258–3270, Bangkok, Thailand. Association for Computational Linguistics.

Hao Yu, Zelan Yang, Shen Li, Yong Li, and Jianxin Wu. 2024. Effectively Compress KV Heads for LLM. *Preprint*, arXiv:2406.07056.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. 2024a. LoRC: Low-Rank Compression for LLMs KV Cache with a Progressive Compression Strategy. *Preprint*, arXiv:2410.03111.

Yike Zhang, Zhiyuan He, Huiqiang Jiang, Chengruidong Zhang, Yuqing Yang, Jianyong Wang, and Lili Qiu. 2025. LeanK: Learnable K Cache Channel Pruning for Efficient Decoding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31122–31137, Suzhou, China. Association for Computational Linguistics.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024b. CaM: Cache Merging for Memory-efficient LLMs Inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 58840–58850, Vienna, Austria. PMLR.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710, New Orleans, LA, USA. Curran Associates, Inc.

Yuxuan Zhu, David H. Yang, Mohammad Mohammadi Amiri, Keerthiram Murugesan, Tejaswini Pedapati, and Pin-Yu Chen. 2026. OjaKV: Context-Aware Online Low-Rank KV Cache Compression. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 10161–10178, San Diego, California, United States. Association for Computational Linguistics.

A Preliminaries

A.1 KV Cache Computation and Update

The inference process of typical LLMs consists of two main stages: *prefilling* and *decoding*. During the *prefilling* stage, the input prompt is passed through all layers of the model to generate the KV cache for each transformer layer. This cache stores the key and value tensors, which are reused in subsequent *decoding* steps, significantly reducing redundant computation. Additionally, during the *decoding* stage, each newly generated token is also projected into key and value tensors, which are appended to the existing KV cache. In this way, the KV cache is progressively updated as the sequence grows, facilitating efficient autoregressive generation.

Prefilling Let $\mathbf{X} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}\times\mathcal{D}}$ denote the input tensor, where $\mathcal{B}$ is the batch size, $\mathcal{T}$ is the prompt length, and $\mathcal{D}$ is the hidden dimension. The corresponding key and value tensors, $\mathbf{S}^K$ and $\mathbf{S}^V$, are computed as follows:

$$\mathbf{S}^{K/V} = \mathbf{X}\mathbf{W}^{K/V}, \tag{25}$$

where $\mathbf{W}^{K/V} \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}'}$ are the key and value projection matrices, respectively. Here, $\mathcal{D}' = h_{kv} \cdot d_h$, with h_{kv} representing the number of KV heads and d_h the head dimension. The computed $\mathbf{S}^{K/V} \in \mathbb{R}^{\mathcal{B}\times\mathcal{T}\times\mathcal{D}'}$ are stored as the initial KV cache to enable efficient retrieval during *decoding*.

Similarly, the query tensor is obtained as

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{Q}, \tag{26}$$

where $\mathbf{W}^{Q} \in \mathbb{R}^{\mathcal{D}\times\mathcal{G}\mathcal{D}'}$. The parameter $\mathcal{G}$ denotes the query-to-KV head ratio. When $\mathcal{G} > 1$, the model corresponds to grouped-query attention (GQA); when $\mathcal{G} = 1$, it reduces to multi-head attention (MHA).

Decoding At each *decoding* timestep t, the newly generated input token $\mathbf{x}_t \in \mathbb{R}^{\mathcal{B}\times 1\times\mathcal{D}}$ is projected to obtain the corresponding key and value vectors $\mathbf{s}_t^{K/V}$:

$$\mathbf{s}_t^{K/V} = \mathbf{x}_t\mathbf{W}^{K/V}. \tag{27}$$

The KV cache is then updated by appending the new entries:

$$\mathbf{S}_t^{K/V} = \mathbf{S}_{t-1}^{K/V} \oplus \mathbf{s}_t^{K/V}, \tag{28}$$

where $\oplus$ denotes concatenation along $\mathcal{T}$.

A.2 Singular Value Decomposition

SVD is a fundamental tool in matrix analysis that enables us to decompose any matrix into a set of orthogonal basis vectors and singular values, facilitating an understanding of its structure and enabling efficient computations such as dimensionality reduction. Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the SVD factorizes it as follows:

$$\mathrm{SVD}(\mathbf{A}) = \mathbf{U}\mathbf{\Sigma}\mathbf{Z}^{\top}, \tag{29}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{Z} \in \mathbb{R}^{n\times n}$ are orthogonal matrices whose columns are, respectively, the left and right singular vectors of $\mathbf{A}$. The matrix $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal (typically rectangular), with non-negative singular values arranged in descending order.

For practical purposes such as dimensionality reduction and compression, one often uses the truncated rank-r SVD, which retains only the r largest singular values and their corresponding singular vectors:

$$\mathrm{SVD}_r(\mathbf{A}) = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{Z}_r^\top, \tag{30}$$

where $\mathbf{U}_r \in \mathbb{R}^{m\times r}$ and $\mathbf{Z}_r \in \mathbb{R}^{n\times r}$ contain the first r left and right singular vectors, respectively, and $\mathbf{\Sigma}_r \in \mathbb{R}^{r\times r}$ is a diagonal matrix of the top r singular values.

B Decoding-Time Complexity Comparison

This section provides a detailed decoding-time complexity comparison between S^{4}R and a full-reconstruction low-rank baseline. We focus on the per-layer cost at a single decoding step and omit the batch dimension for clarity. Let $\mathcal{L}_t$ denote the current non-sink cache length, s the number of sink tokens, $\mathcal{D}'$ the KV hidden dimension, r the low-rank dimension, $\mathcal{G}$ the query-to-KV head ratio, $\mathcal{W}$ the query-window size, and ρ the sparse reconstruction ratio. The number of non-sink tokens reconstructed by S^{4}R is

$$m_t = |\widehat{\mathcal{I}}| = \max(\mathcal{W}, \lfloor \rho\mathcal{L}_t \rfloor). \tag{31}$$

For long contexts where $\rho\mathcal{L}_t \geq \mathcal{W}$, we have $m_t \approx \rho\mathcal{L}_t$.

B.1 Full-Reconstruction Low-Rank Decoding

A natural alternative to sparse reconstruction is to store the non-sink KV cache in coefficient form but reconstruct the entire cache at every decoding step before applying attention. In this full-reconstruction baseline, all cached non-sink keys and values are recovered as

$$\mathbf{K}_t = \mathbf{C}_t^K(\mathbf{B}^K)^\top, \qquad \mathbf{V}_t = \mathbf{C}_t^V(\mathbf{B}^V)^\top. \tag{32}$$

The reconstruction cost for keys and values is therefore

$$O(2\mathcal{L}_t r\mathcal{D}'), \tag{33}$$

which we write as $O(\mathcal{L}_t r\mathcal{D}')$ after dropping constant factors. The final attention is then computed over the full non-sink cache, together with the sink tokens. Ignoring the small sink-token term for long contexts, the attention score computation and value aggregation have cost

$$O(\mathcal{G}\mathcal{L}_t\mathcal{D}'). \tag{34}$$

Thus, the dominant per-step decoding complexity of full reconstruction is

$$\begin{aligned} C_{\text{full-rec}} &= O\left(\mathcal{L}_t r\mathcal{D}' + \mathcal{G}\mathcal{L}_t\mathcal{D}'\right) \\ &= O\left(\mathcal{L}_t\mathcal{D}'(r+\mathcal{G})\right). \end{aligned} \tag{35}$$

This baseline reduces persistent KV memory because it stores low-rank coefficients, but its decoding computation still scales with the full cache length in both reconstruction and attention.

B.2 S^{4}R Sparse-Reconstruction Decoding

S^{4}R avoids reconstructing the full cache. At each decoding step, the new query, key, and value states are first projected into the learned low-rank subspaces. The cost of this cache update is

$$O((\mathcal{G}+2)\mathcal{D}'r), \tag{36}$$

where the $\mathcal{G}$ term comes from grouped query projection and the remaining two terms come from key and value projection. This term does not scale with $\mathcal{L}_t$ and is therefore lower order for long contexts.

S^{4}R then estimates token relevance in the latent space by multiplying the query-coefficient window with the cached key coefficients:

$$\mathbf{Z} = \frac{\mathbf{C}^Q(\mathbf{C}_t^K)^\top}{\sqrt{r}}, \qquad \mathbf{Z} \in \mathbb{R}^{\mathcal{G}\times\mathcal{W}\times\mathcal{L}_t}. \tag{37}$$

This latent scoring step costs

$$O(\mathcal{G}\mathcal{W}\mathcal{L}_t r). \tag{38}$$

The subsequent softmax, pooling, group aggregation, and TopK selection add lower-order terms such as $O(\mathcal{G}\mathcal{W}\mathcal{L}_t)$ and $O(\mathcal{L}_t \log k)$, or approximately $O(\mathcal{L}_t)$ with efficient selection.

After token selection, S^{4}R reconstructs only m_t selected non-sink tokens:

$$\widehat{\mathbf{K}} = \mathbf{C}^K_{t\,[\widehat{\mathcal{I}}]}(\mathbf{B}^K)^\top, \qquad \widehat{\mathbf{V}} = \mathbf{C}^V_{t\,[\widehat{\mathcal{I}}]}(\mathbf{B}^V)^\top. \tag{39}$$

The sparse reconstruction cost is

$$O(2m_t r\mathcal{D}'), \tag{40}$$

which is $O(m_t r\mathcal{D}')$ after omitting constant factors. The final exact attention is performed only

over the reconstructed non-sink tokens and the full-precision sink tokens, with cost

$$O(\mathcal{G}(s+m_t)\mathcal{D}'). \tag{41}$$

Ignoring lower-order terms and the small sink-token contribution, the dominant S^{4}R decoding complexity is therefore

$$C_{\mathrm{S^4R}} = O\left(\mathcal{G}\mathcal{W}\mathcal{L}_t r + m_t r\mathcal{D}' + \mathcal{G}m_t\mathcal{D}'\right). \tag{42}$$

When $m_t \approx \rho\mathcal{L}_t$, this becomes

$$C_{\mathrm{S^4R}} = O\left(\mathcal{G}\mathcal{W}\mathcal{L}_t r + \rho\mathcal{L}_t r\mathcal{D}' + \rho\mathcal{G}\mathcal{L}_t\mathcal{D}'\right). \tag{43}$$

Equivalently,

$$C_{\mathrm{S^4R}} = O\left(\mathcal{L}_t\left[\mathcal{G}\mathcal{W}r + \rho\mathcal{D}'(r+\mathcal{G})\right]\right). \tag{44}$$

B.3 Asymptotic Comparison

Equations (35) and (44) show the main difference between the two approaches. Full reconstruction pays the reconstruction and attention cost over all $\mathcal{L}_t$ cached non-sink tokens; S^{4}R reduces both reconstruction and exact attention from $\mathcal{L}_t$ tokens to approximately $\rho\mathcal{L}_t$ tokens, but introduces latent-space relevance scoring. Therefore, S^{4}R is computationally favorable over full reconstruction when

$$\mathcal{G}\mathcal{W}r + \rho\mathcal{D}'(r+\mathcal{G}) < \mathcal{D}'(r+\mathcal{G}). \tag{45}$$

Equivalently,

$$\mathcal{G}\mathcal{W}r < (1-\rho)\mathcal{D}'(r+\mathcal{G}). \tag{46}$$

This condition is usually satisfied when the query window $\mathcal{W}$ and rank r are much smaller than the original KV dimension $\mathcal{D}'$, and when the reconstruction ratio ρ is substantially below one.

B.4 Concrete Example

We instantiate the above comparison using the latency-oriented efficiency setting in Table 3, where $\mathcal{W} = 4$. Consider $r = 204$, $\rho = 0.2$, and $\mathcal{D}' = 1024$. After removing the common factor $\mathcal{L}_t$, the full-reconstruction cost coefficient is

$$\begin{aligned} r\mathcal{D}' + \mathcal{G}\mathcal{D}' &= 204 \times 1024 + 1024\mathcal{G} \\ &= 208{,}896 + 1024\mathcal{G}. \end{aligned} \tag{47}$$

For S^{4}R, the latent-scoring coefficient is

$$\mathcal{G}\mathcal{W}r = 4 \times 204\mathcal{G} = 816\mathcal{G}, \tag{48}$$

while the sparse reconstruction coefficient is

$$\rho r\mathcal{D}' = 0.2 \times 204 \times 1024 = 41{,}779.2, \tag{49}$$

and the sparse attention coefficient is

$$\rho\mathcal{G}\mathcal{D}' = 0.2 \times 1024\mathcal{G} = 204.8\mathcal{G}. \tag{50}$$

Thus,

$$C_{\mathrm{S^4R}}/\mathcal{L}_t \approx 41{,}779 + 1021\mathcal{G}, \tag{51}$$

whereas

$$C_{\text{full-rec}}/\mathcal{L}_t \approx 208{,}896 + 1024\mathcal{G}. \tag{52}$$

For $\mathcal{G} = 1$, this gives approximately

$$\frac{C_{\mathrm{S^4R}}}{C_{\text{full-rec}}} \approx \frac{42{,}800}{209{,}920} \approx 0.204. \tag{53}$$

For $\mathcal{G} = 4$, the ratio is

$$\frac{C_{\mathrm{S^4R}}}{C_{\text{full-rec}}} \approx \frac{45{,}862}{212{,}922} \approx 0.215. \tag{54}$$

These estimates indicate that, under this setting, S^{4}R reduces the dominant reconstruction-and-attention arithmetic by roughly 4.6–5$\times$ relative to full reconstruction. The latent scoring term is small compared with the saved reconstruction cost because

$$\mathcal{W}r = 4 \times 204 = 816 \ll \rho r\mathcal{D}' = 41{,}779.2. \tag{55}$$

Consequently, the leading effect is that sparse reconstruction replaces the full reconstruction cost $O(\mathcal{L}_t r\mathcal{D}')$ with $O(\rho\mathcal{L}_t r\mathcal{D}')$, yielding an approximate $1/\rho$ reconstruction-compute reduction when the latent-scoring overhead is small.

C More Related Work

The main text positions S^{4}R within low-rank KV-cache compression. This appendix covers complementary compression axes and then isolates the operational differences between S^{4}R and its nearest alternatives.

C.1 Complementary KV-Cache Compression Axes

Sequence reduction and query-aware retrieval. Eviction methods reduce persistent memory by retaining only a subset of past positions. StreamingLLM combines attention sinks with a sliding window, while H2O and Scissorhands retain tokens using accumulated or persistent importance signals (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2023). Later methods allocate the token budget more selectively: NACL combines local

and global importance, Ada-KV distributes cache budgets across heads, SnapKV uses recent prompt attention, and PyramidInfer and PyramidKV vary retention across layers (Chen et al., 2024; Feng et al., 2025; Li et al., 2024; Yang et al., 2024; Cai et al., 2025). Quest and DuoAttention make access query- or head-dependent rather than applying a uniform retention policy (Tang et al., 2024; Xiao et al., 2025). RocketKV combines coarse permanent eviction with fine-grained top-k sparse attention, using both head- and sequence-dimensional approximations during selection (Behnam et al., 2025). These approaches principally reduce the sequence dimension or the set of KV entries accessed by attention. S^{4}R instead keeps a coefficient representation for every non-sink position and makes the full-dimensional reconstruction set query-dependent, so positions omitted at one decoding step remain recoverable later.

Token- and layer-level merging. Merging methods preserve information from removed entries by combining them with retained representatives. CaM merges to-be-evicted states according to attention-derived importance, KVMerger identifies groups of similar states, Chelsea performs online clustering along the sequence dimension, and GRKV distributes information through a global regression objective (Zhang et al., 2024b; Wang et al., 2024c, 2025a; Peng et al., 2026). MiniCache exploits a different redundancy axis by merging KV states across adjacent middle-to-deep layers (Liu et al., 2024a). These methods reduce the number of independently stored states across tokens or layers. S^{4}R does not merge token identities or layer states: it stores per-position coefficients in layer-specific key/value subspaces and reconstructs selected positions when needed. Token/layer merging and coefficient-space compression are therefore distinct and potentially composable, but they impose different information-loss mechanisms.

Quantization and channel pruning. Quantization reduces the bit width of cached states. KIVI uses asymmetric low-bit schemes for keys and values, KVQuant combines per-channel and non-uniform quantization with explicit outlier handling, and GEAR approximates quantization error with low-rank and sparse residual components (Liu et al., 2024b; Hooper et al., 2024; Kang et al., 2024). TaDA further adapts precision across layers with mean-centering (Joshi et al., 2025). LeanK targets the channel dimension directly by learning static sparsity for the key cache (Zhang et al., 2025). In contrast, S^{4}R represents both keys and values through prompt-derived low-rank bases and coefficient caches. Bit-width reduction or channel pruning could be applied to those coefficients, but such combinations would require evaluating whether quantization or pruning distorts latent-space relevance scores as well as reconstructed attention.

Memory hierarchy and cache reuse. System-oriented methods change where compressed information resides or how it is reused. KV-Compress couples block eviction with a PagedAttention-style memory manager, KVzip builds query-agnostic compressed caches reusable across future queries, and SpeCache keeps the full cache in CPU memory while using a compact GPU-side representation to guide fetching (Rehg, 2024; Kim et al., 2025; Jie et al., 2025). These methods address allocation, reuse, and data movement, whereas S^{4}R changes the mathematical representation stored for each prompt. Applying such placement or offloading policies to a coefficient cache is a systems-level extension rather than a competing subspace-construction method.

C.2 Fine-Grained Positioning of S^{4}R

Where and when the low-rank representation is constructed. Palu and LoRC determine their factorized representations before serving the current prompt, avoiding prompt-time decomposition but not adapting the subspace to the realized context (Chang et al., 2025; Zhang et al., 2024a). ShadowKV and xKV construct prompt-dependent representations and are therefore closer to S^{4}R (Sun et al., 2025; Chang et al., 2026). ShadowKV decomposes current-prompt pre-RoPE keys, keeps a low-rank key representation on GPU, and retrieves selected values from an offloaded cache; xKV performs prompt-time cross-layer SVD and shares aligned low-rank structure across grouped layers. S^{4}R instead estimates separate key and value bases from a sampled subset of non-sink prompt tokens, then stores coefficients for all non-sink positions. The intended trade-off is narrower: it preserves prompt dependence while reducing the amount of prompt data participating in decomposition. Our efficiency measurements, rather than the method definitions alone, establish the resulting TTFT and generation-latency differences.

Persistent compression versus sparse access. StreamingLLM and SnapKV permanently retain a

selected subset of full-dimensional positions, while Quest and Loki primarily reduce attention work through query-dependent sparse access (Xiao et al., 2024; Li et al., 2024; Tang et al., 2024; Singhania et al., 2024). RocketKV explicitly combines permanent eviction with a second sparse-attention stage (Behnam et al., 2025). S^{4}R couples these concerns differently: its persistent memory reduction comes from coefficient-space storage, whereas latent scoring determines only which positions are reconstructed for the current attention step. This separation lets a position remain cheaply recoverable after it is skipped, at the cost of storing coefficients for the full non-sink sequence and performing latent scoring during decoding.

Takeaway. Across these axes, the distinctive combination in S^{4}R is prompt-sampled construction of both key and value subspaces, persistent per-position coefficient storage, full-precision sink preservation, and query-dependent sparse reconstruction. Fixed low-rank methods remove most prompt-time adaptation; prompt-wide methods retain adaptation but pay decomposition or update costs over more prompt states; eviction and retrieval methods reduce the active sequence without necessarily compressing every retained state; and quantization, merging, and systems techniques target complementary representation or deployment dimensions.

$\mathcal{W}$	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
1	46.82	20.53	21.13	19.41	4.15	25.90
4	52.60	19.99	22.60	23.56	3.95	28.46
8	52.71	20.13	22.75	23.41	3.95	28.56
16	53.32	20.22	22.26	24.25	3.86	28.68
32	51.29	20.69	22.06	23.43	3.39	28.09
64	48.82	20.60	21.80	23.84	3.72	27.38
Qwen3-4B						
1	64.83	37.87	37.40	5.08	95.00	47.31
4	65.05	38.37	37.98	4.99	95.83	47.73
8	64.99	38.55	37.81	5.01	95.33	47.67
16	65.79	38.56	37.63	5.03	95.33	47.85
32	65.40	38.53	37.56	5.05	96.17	47.79
64	65.30	38.11	37.56	5.11	95.00	47.55
Llama-3.1-8B						
1	61.62	41.13	44.64	29.49	99.50	51.92
4	63.60	42.38	46.33	32.06	99.50	53.50
8	62.99	42.74	45.98	32.14	99.50	53.34
16	62.16	42.75	46.08	32.22	99.50	53.15
32	61.92	42.32	46.28	32.29	99.50	53.03
64	58.75	42.03	46.24	32.46	99.50	52.09
Qwen2.5-14B						
1	66.16	38.05	52.16	19.38	92.50	52.82
4	67.01	38.91	51.99	19.97	94.50	53.47
8	66.88	38.98	52.59	20.19	94.50	53.64
16	67.19	38.65	52.69	20.29	94.50	53.67
32	66.71	38.20	51.88	20.25	94.50	53.19
64	64.08	38.85	51.51	20.45	93.50	52.48

Table 7: Effect of the query-window size $\mathcal{W} \in \{1, 4, 8, 16, 32, 64\}$ on grouped LongBench performance. All other S^{4}R hyperparameters are fixed, and shaded rows denote the default setting used in the main experiments.

D More Ablation Studies

D.1 Effect of Query Window

S^{4}R uses a sliding query window of length $\mathcal{W}$ for two related purposes during decoding. First, the projected queries in this window are used to estimate token relevance in the latent coefficient space, where attention scores are averaged across recent queries before selecting globally important positions. Second, the same window determines the local tokens that are always reconstructed and attended to in full precision. Thus, $\mathcal{W}$ controls how much short-range decoding context is used for stable token scoring and how much recent context is guaranteed to be preserved in the sparse reconstruction set.

This ablation varies $\mathcal{W}$ while keeping the rank, sampling strategy, reconstruction ratio, and pooling configuration fixed. When $\mathcal{W} = 1$, token selection is driven only by the current query, which provides the lowest query-coefficient storage and latent scoring cost but may produce noisier global-token estimates. Larger windows aggregate relevance over multiple recent queries and preserve more local tokens, which can improve robustness for tasks where the answer depends on a short span of recent decoding context. However, increasing $\mathcal{W}$ also increases the query-coefficient buffer size and the latent attention cost, and it allocates more of the reconstruction budget to local tokens, leaving fewer slots for globally selected positions when the total reconstruction ratio is fixed.

Table 7 compares $\mathcal{W} \in \{1, 4, 8, 16, 32, 64\}$ across LongBench task groups. The results show a clear preference for moderate query windows. Using only the current query is consistently weaker: increasing $\mathcal{W}$ from 1 to the best-performing setting improves the average score from 25.90 to 28.68 on Llama-3.2-1B, from 47.31 to 47.85 on Qwen3-4B, from 51.92 to 53.50 on Llama-3.1-8B, and from 52.82 to 53.67 on Qwen2.5-14B. These gains indicate that aggregating relevance over several recent queries produces more stable global-token

$r^K : r^V$	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
1 : 1	51.29	20.69	22.06	23.43	3.39	28.09
1 : 1.5	51.24	20.46	22.15	23.44	4.88	28.17
1 : 2	50.17	20.33	22.87	24.33	5.06	28.14
Qwen3-4B						
1 : 1	65.40	38.53	37.56	5.05	96.17	47.79
1 : 1.5	64.62	39.00	38.81	5.10	96.50	48.08
1 : 2	64.74	38.70	38.57	5.10	96.83	48.00
Llama-3.1-8B						
1 : 1	61.92	42.32	46.28	32.29	99.50	53.03
1 : 1.5	60.32	43.42	46.34	32.07	99.50	52.89
1 : 2	59.96	43.13	45.98	32.39	99.50	52.65
Qwen2.5-14B						
1 : 1	66.71	38.20	51.88	20.25	94.50	53.19
1 : 1.5	67.03	39.94	52.29	20.67	96.50	54.09
1 : 2	67.13	39.34	52.02	20.98	95.50	53.81

Table 8: Effect of key–value rank allocation under a fixed total coefficient budget. The ratio $r^K : r^V$ specifies the key and value ranks, and shaded rows denote the balanced default allocation.

estimates than single-query selection.

The benefit, however, saturates quickly. The strongest averages are obtained by $\mathcal{W} = 16$ for Llama-3.2-1B, Qwen3-4B, and Qwen2.5-14B, and by $\mathcal{W} = 4$ for Llama-3.1-8B. The default $\mathcal{W} = 32$ remains close to the optimum, trailing the best setting by 0.59, 0.06, 0.47, and 0.48 points on the four models, respectively, while maintaining a larger guaranteed local reconstruction window. Increasing the window further to $\mathcal{W} = 64$ lowers the average score for all models compared with $\mathcal{W} = 32$. This suggests that overly large query windows dilute current-step relevance and reserve more reconstruction budget for local tokens, thereby reducing the coverage of globally important positions. We therefore keep $\mathcal{W} = 32$ as the default in the main experiments not because it is always the best setting in this ablation, but because it provides near-optimal accuracy and matches the default window size used by SnapKV.

D.2 Effect of Different Rank Allocation

Prior work observes that pre-RoPE keys exhibit stronger low-rank characteristics than values (Lee et al., 2025; Sun et al., 2025). This asymmetry suggests that keys may tolerate a smaller latent rank, while values may require a larger rank to preserve the information aggregated into the final attention output. Motivated by this observation, we restrict this ablation to settings where the key rank is no larger than the value rank, i.e., $r^K \leq r^V$. The goal is to test whether shifting part of the coefficient budget from keys to values improves reconstruction quality without substantially degrading latent-space token scoring.

To study this trade-off, we compare several key-to-value rank ratios while keeping the total coefficient budget fixed. Concretely, for a target budget corresponding to the default rank r, we choose (r^K, r^V) such that $r^K + r^V = 2r$ and vary the allocation ratio $r^K : r^V$.

Table 8 reports the results across LongBench task groups. The results show that asymmetric rank allocation can be beneficial, but the effect depends on model scale. On Llama-3.2-1B, the average scores are nearly tied, with 1:1.5 slightly outperforming 1:1 by 0.08 points and 1:2 by 0.03 points. On Qwen3-4B, allocating more rank to values improves the average score from 47.79 to 48.08 for 1:1.5 and 48.00 for 1:2. The same trend is clearer on Qwen2.5-14B, where 1:1.5 improves the average score from 53.19 to 54.09 and 1:2 reaches 53.81. In contrast, Llama-3.1-8B favors the balanced allocation, with 1:1 achieving 53.03 compared with 52.89 and 52.65 for 1:1.5 and 1:2. Overall, these results support the motivation that values can require more rank than pre-RoPE keys, but an overly value-heavy allocation is not uniformly optimal. We therefore keep the balanced allocation as the default in the main experiments for simplicity and robustness, while noting that a mildly value-biased allocation such as 1:1.5 can further improve some model families.

D.3 Effect of Different Recent-to-Uniform Sampling Ratios

S^{4}R constructs the prompt-aware key/value bases from a sampled subset of non-sink tokens. The sampling policy combines a contiguous block of recent tokens with uniformly sampled earlier tokens. These two sources are complementary: recent tokens tend to be more relevant to the immediate decoding state and better match the local distribution near generation, while uniformly sampled tokens provide broader coverage of the full prompt and reduce the risk of missing globally important context. The default configuration uses a recent-to-uniform split of $7 : 3$.

Table 9 compares several ratios while keeping the total number of sampled tokens fixed. The results show that S^{4}R is relatively robust to moderate changes in the split, but combining recent and uniform samples is generally preferable to using only

Ratio	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
7 : 3	51.29	20.69	22.06	23.43	3.39	28.09
6 : 4	52.09	20.67	22.59	22.94	3.85	28.44
5 : 5	50.46	20.17	21.56	23.24	4.25	27.64
4 : 6	50.52	20.69	21.76	24.05	3.94	27.90
10 : 0	50.80	19.74	22.16	24.09	3.88	27.82
Qwen3-4B						
7 : 3	65.40	38.53	37.56	5.05	96.17	47.79
6 : 4	64.86	38.41	37.82	5.11	90.42	47.16
5 : 5	66.15	38.41	38.01	5.13	93.00	47.80
4 : 6	65.19	38.04	37.88	5.04	93.50	47.44
10 : 0	64.95	36.91	37.32	5.10	96.33	47.18
Llama-3.1-8B						
7 : 3	61.92	42.32	46.28	32.29	99.50	53.03
6 : 4	60.96	42.75	45.81	31.96	100.00	52.77
5 : 5	60.92	41.98	45.36	32.48	100.00	52.48
4 : 6	60.09	43.12	46.19	32.82	100.00	52.82
10 : 0	61.91	41.24	45.52	32.44	100.00	52.59
Qwen2.5-14B						
7 : 3	66.71	38.20	51.88	20.25	94.50	53.19
6 : 4	66.26	39.36	52.56	20.25	94.50	53.57
5 : 5	66.53	39.75	52.44	20.05	95.00	53.75
4 : 6	66.95	40.17	52.64	20.52	91.50	53.75
10 : 0	66.89	38.12	51.08	20.36	94.25	52.99

Table 9: Effect of the recent-to-uniform sampling ratio used to construct prompt-aware key/value bases. The total number of sampled tokens is fixed, and shaded rows denote the default 7:3 split.

recent tokens. For Llama-3.2-1B, $6:4$ obtains the best average score of 28.44, while the default $7:3$ remains close at 28.09. For Qwen3-4B, $5:5$ and $7:3$ are nearly tied, with average scores of 47.80 and 47.79, respectively; however, $7:3$ performs better on passage retrieval than $5:5$. For Llama-3.1-8B, the default $7:3$ achieves the best average score of 53.03, outperforming the other ratios by 0.21–0.55 points. For Qwen2.5-14B, more uniform coverage is beneficial: $5:5$ and $4:6$ both reach 53.75, compared with 53.19 for $7:3$. In contrast, the recent-only $10:0$ setting is not consistently competitive, suggesting that relying only on the most recent sampled tokens can miss useful global prompt structure. Overall, the results indicate that no single split is universally optimal across all models, but hybrid recent-plus-uniform sampling is important. We keep $7:3$ as the default because it is consistently strong, especially on Llama models and passage retrieval, while maintaining a simple fixed sampling rule.

D.4 Effect of Reconstruction Ratio

The reconstruction ratio ρ controls how many non-sink positions are reconstructed and attended to in full precision at each decoding step. In the main experiments, we set ρ to match the target compression ratio η, so the sparse reconstruction budget scales with the memory budget used by each model setting. This ablation tests whether S^4R can use a smaller reconstruction set during decoding while keeping the compressed coefficient cache and other hyperparameters unchanged. Reducing ρ directly lowers the number of reconstructed key/value vectors and the size of the exact attention set, but it may also remove globally relevant positions from the final sparse attention computation.

ρ	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
0.2	51.29	20.69	22.06	23.43	3.39	28.09
0.15	50.50	19.83	22.75	24.00	3.63	27.90
Qwen3-4B						
0.2	65.40	38.53	37.56	5.05	96.17	47.79
0.15	64.69	38.38	37.83	5.03	94.33	47.46
Llama-3.1-8B						
0.2	61.92	42.32	46.28	32.29	99.50	53.03
0.15	59.40	42.08	45.78	32.06	99.00	52.08
Qwen2.5-14B						
0.3	66.71	38.20	51.88	20.25	94.50	53.19
0.15	64.10	38.13	51.48	18.66	93.50	52.12

Table 10: Effect of the reconstruction ratio ρ during sparse decoding. The coefficient cache and other hyperparameters are unchanged, and shaded rows denote the default ρ for each model setting.

Table 10 compares the default reconstruction ratio with a smaller setting of $\rho = 0.15$. The results show that S^4R is reasonably robust to reconstructing fewer tokens, but the default setting remains consistently stronger. On Llama-3.2-1B, reducing ρ from 0.2 to 0.15 only decreases the average score from 28.09 to 27.90, and even improves MDQA, LCC, and PRE slightly. On Qwen3-4B, the average score drops by 0.33 points, mainly due to lower Fewshot and PRE scores. The larger Llama-3.1-8B and Qwen2.5-14B settings are more sensitive: their average scores decrease by 0.95 and 1.07 points, respectively. This suggests that a smaller reconstruction set can preserve most of the performance when further decoding-time savings are desired, but the default $\rho = \eta$ provides a safer accuracy–efficiency trade-off, especially for stronger models whose outputs rely more on recovering a broader set of relevant context positions.

D.5 Effect of Layer-wise Full Reconstruction

SALS applies full reconstruction to selected layers based on its layer-wise analysis, specifically the first two layers and the last layer, to reduce the approximation error in layers that are more sensitive to compression (Mu et al., 2025). Motivated by this design, we test whether a similar layer-wise full-reconstruction strategy is also beneficial for S^4R. In this ablation, "w/" denotes applying full reconstruction to these selected layers, while "w/o" denotes the default S^4R configuration that uses sparse reconstruction consistently across layers. All other hyperparameters are kept unchanged.

Table 11 shows that layer-wise full reconstruction does not consistently improve S^4R. On Llama-3.2-1B, it improves the average score from 28.09 to 28.45, mainly due to gains on Fewshot, MDQA, LCC, and PRE. However, the same strategy reduces the average score on the other three models: from 47.79 to 47.45 on Qwen3-4B, from 53.03 to 52.33 on Llama-3.1-8B, and from 53.19 to 52.94 on Qwen2.5-14B. The drops are especially visible in MDQA and PRE for Qwen3-4B and in Fewshot and MDQA for Llama-3.1-8B.

These results suggest that the layer-wise full-reconstruction heuristic used by SALS is not directly transferable to S^4R. S^4R already preserves sink tokens and reconstructs a query-dependent sparse set at every layer; forcing full reconstruction on fixed layers can change the balance between exact local/global token recovery and the prompt-adaptive sparse reconstruction mechanism. It also introduces additional decoding-time reconstruction cost. Since the accuracy gains are limited to the smallest model and are not robust across model families, we do not use layer-wise full reconstruction as the default.

FR	Fewshot	SDQA	MDQA	LCC	PRE	Avg.
Llama-3.2-1B						
w/o	51.29	20.69	22.06	23.43	3.39	28.09
w/	52.29	20.04	22.74	23.99	3.77	28.45
Qwen3-4B						
w/o	65.40	38.53	37.56	5.05	96.17	47.79
w/	65.62	38.48	36.52	5.03	95.08	47.45
Llama-3.1-8B						
w/o	61.92	42.32	46.28	32.29	99.50	53.03
w/	60.27	42.22	45.64	32.20	99.00	52.33
Qwen2.5-14B						
w/o	66.71	38.20	51.88	20.25	94.50	53.19
w/	66.26	37.82	52.03	19.97	94.00	52.94

Table 11: Effect of applying layer-wise full reconstruction to selected layers. "w/" applies full reconstruction to the first two layers and the last layer, whereas "w/o" is the default sparse-reconstruction setting; shaded rows denote the default.

E Implementation Details

E.1 LongBench

In this section, we provide the detailed statistics of the LongBench benchmark (Bai et al., 2024) used in our evaluation. As shown in Table 12, the selected datasets cover five major categories: single-document QA, multi-document QA, few-shot learning, synthetic tasks, and code completion. These tasks represent typical long-context scenarios with average input lengths ranging from approximately 1.2k to over 18k words. We also specify the decoding lengths and the primary evaluation metrics (e.g., F1 score, Rouge-L, and Exact Match) for each task to provide a comprehensive reference for our experimental setup in Section 4.1.

We evaluate our method on the LongBench benchmark using Llama-3.2-1B-Instruct, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen3-4B-Instruct-2507 (Yang et al., 2025a) and Qwen2.5-14B-Instruct (Qwen et al., 2025). Llama-3.2-1B-Instruct is a 1-billion-parameter model with a 128k-token context window, while Llama-3.1-8B-Instruct is an 8-billion-parameter model with a 128k-token context window. Qwen3-4B-Instruct-2507 is a 4-billion-parameter model with a 32k-token context window natively and 128k tokens with YaRN (Peng et al., 2024), while Qwen2.5-14B-Instruct is a 14-billion-parameter model with a 32k-token context window. For LongBench samples exceeding 32k tokens (e.g., in NarrativeQA), we truncate the input to the rightmost 32k tokens to fit within the Qwen2.5-14B-Instruct's constraints.

E.2 RULER

RULER (Hsieh et al., 2024) is built for maximal flexibility, supporting a vast range of sequence lengths and task complexities. Each task allows for a combinatorial number of configuration options, as summarized in Table 13. In our primary experiments, we assess Qwen2.5-7B-Instruct-1M (Yang et al., 2025b) on seven representative tasks spanning RULER's two primary categories: retrieval and multi-hop tracing. For each task, we prepare samples of 128k tokens and generate 20 samples to assess performance. Unless otherwise specified, the decoding length is set to 32 tokens.

Dataset	Source	Avg len	Decoding len	Metric	Language	#data
Single-Document QA						
NarrativeQA	Literature, Film	18,409	128	F1	English	200
Qasper	Science	3,619	128	F1	English	200
MultiFieldQA-en	Multi-field	4,559	64	F1	English	150
Multi-Document QA						
HotpotQA	Wikipedia	9,151	32	F1	English	200
2WikiMultihopQA	Wikipedia	4,887	32	F1	English	200
MuSiQue	Wikipedia	11,214	32	F1	English	200
Few-shot Learning						
TREC	Web question	5,177	64	Accuracy (CLS)	English	200
TriviaQA	Wikipedia, Web	8,209	32	F1	English	200
SAMSum	Dialogue	6,258	128	Rouge-L	English	200
Synthetic Task						
PassageRetrieval-en	Wikipedia	9,289	32	Accuracy (EM)	English	200
Code Completion						
LCC	Github	1,235	64	Edit Sim	Python/C#/Java	500

Table 12: Summary of LongBench dataset statistics. 'Avg len' denotes average input length (number of words for English/code datasets). 'Decoding len' is the number of tokens in the decoding sequence. 'Accuracy (CLS)' indicates classification accuracy, and 'Accuracy (EM)' stands for exact match accuracy.

Task	Configurations		
	Subtask-1	Subtask-2	Subtask-3
MK-NIAH	num_keys = 2 type_key = word type_value = number type_haystack = essay	type_key = word type_value = number type_haystack = essay **line retrieval**	type_key = word type_value = uuid type_haystack = essay **KV retrieval**
MV-NIAH	num_values = 2, type_key = word, type_value = number, type_haystack = essay		
MQ-NIAH	num_queries = 2, type_key = word, type_value = number, type_haystack = essay		
VT	num_chains = 1, num_hops = 4		
FWE	$\alpha = 2.0$		

Table 13: Seven task configurations in RULER.

Both LongBench and RULER are evaluated using LM-Evaluation-Harness (Gao et al., 2024).