OjaKV: Context-Aware Online Low-Rank KV Cache Compression

Anonymous ACL submission

Abstract

The expanding long-context capabilities of large language models are constrained by a significant memory bottleneck: the key-value (KV) cache required for autoregressive generation. This bottleneck is substantial; for instance, a Llama-3.1-8B model processing a 32K-token prompt at a batch size of 4 requires approximately 16 GB for its KV cache, a size exceeding the model's weights. While KV-cache compression via low-rank projection is a promising direction, existing methods' rely on a static, offline-learned subspace that performs poorly under data distribution shifts. To overcome these limitations, we introduce **OjaKV**, a novel framework that integrates a strategic hybrid storage policy with online subspace adaptation. First, OjaKV recognizes that not all tokens are equally important for compression; it preserves the crucial tokens in full-rank, maintaining high-fidelity anchors for attention. Second, for the vast majority of intermediate tokens, it applies low-rank compression by incrementally adapting the projection basis using Oja's algorithm for online principal component analysis. This adaptation involves a comprehensive update during prompt prefilling and lightweight periodic updates during decoding, ensuring the subspace remains aligned with the evolving context. Crucially, our framework is fully compatible with modern attention modules like *FlashAttention*. Experiments demonstrate that OjaKV maintains or even improves zero-shot accuracy at high compression ratios. In particular, OjaKV achieves its strongest gains on very long-context benchmarks that require complex reasoning, highlighting the importance of online subspace adaptation in dynamically tracking context shifts. Furthermore, our approach is compatible with token-selection methods, enabling compounded memory savings. These results establish our hybrid framework as a practical, plug-and-play solution for memory-efficient long-context inference without requiring model fine-tuning.

1 Introduction

Large language models (LLMs) such as GPT-4o (OpenAI et al., 2024) and Deepseek-R1 (DeepSeek-AI et al., 2025) have demonstrated remarkable performance across diverse domains, including coding (Nam et al., 2024), mathematics (Setlur et al., 2024), and open-ended text generation (Kumichev et al., 2024). However, as model capabilities and context length expand, GPU memory emerges as a critical bottleneck for inference. The memory footprint arises from two primary sources: (i) model weights, with a model like Llama-3.1-8B requiring 16 GB alone; and (ii) the Key-Value (KV) cache used during prompt prefilling and autoregressive decoding. For instance, processing a 32K-token prompt with Llama-3.1-8B in float16 precision at a batch size of 4 consumes an additional 16 GB for the KV cache, rivalling the size of the model weights themselves. This substantial memory consumption makes long-context inference prohibitive on all but high-end hardware.

To mitigate this challenge, a variety of methods have been proposed to optimize KV-cache memory usage (Shi et al., 2024). These approaches can be grouped into four categories: (1) *Quantization*, which stores keys and values at a lower precision (e.g., 8-bit) (Hooper et al., 2024; Liu et al., 2024c); (2) *Token Selection*, which prunes or merges tokens deemed unimportant based on attention scores or heuristic saliency measures (Xiao et al., 2023; Li et al., 2024; Zhang et al., 2023); (3) *Offloading*, which transfers the KV cache to CPU memory and selectively streams it back during decoding (Tang et al., 2024; Sun et al., 2024; Zhu et al., 2025); and (4) *Low-rank Approximation*, which projects keys and values into a lower-dimensional subspace (Saxena et al., 2024; Lin et al., 2024).

Our work focuses on this fourth direction. By compressing each key and value vector from di-

mension d (e.g., $d = 128$) to r (e.g., $r = 96$), low-rank methods can reduce cache memory by $((1 - r/d) \times 100)\%$ while preserving model accuracy, yielding substantial savings for long-context inference. While token selection has become a widely adopted strategy, we provide theoretical support showing that low-rank projection and token eviction are compatible, offering multiplicative benefits that enable even greater memory reductions when combined.

Existing low-rank methods fall into two main categories. (1) *Weight-decomposition* techniques directly factorize the linear projection weights for query, key, and value ($\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$) into low-rank matrices, thereby caching already-compressed intermediate states (Chang et al., 2024). However, this approach often incurs a noticeable degradation in accuracy. (2) *Projection-based* techniques learn fixed orthonormal projeciton bases ($\boldsymbol{U}_q$, $\boldsymbol{U}_k$, $\boldsymbol{U}_v$) from a calibration dataset. These bases are then used to compress the KV cache, which is reconstructed during attention computation (Saxena et al., 2024; Lin et al., 2024). While effective, these static bases implicitly assume that inference prompts will follow the same distribution as the calibration data. In practice, distribution shifts (e.g., from dialogue to code generation) cause the approximation to deteriorate, harming generation quality.

To address these limitations, we propose **OjaKV**, a novel framework for KV-cache compression that operates on two core principles. Our first key insight is that uniform compression across all tokens is suboptimal. Motivated by the findings of attention sinks (Xiao et al., 2023) and SnapKV (Li et al., 2024), OjaKV employs a hybrid storage policy that strategically excludes the crucial tokens with high reconstruction error from low-rank projection. This preserves their full-rank fidelity, creating stable anchors for the attention mechanism and forming a significantly stronger performance baseline. Second, for the remaining intermediate tokens, we incorporate online subspace adaptation using Oja's incremental principle component analysis (PCA) (Oja, 1997). This mechanism performs a comprehensive update during the prefill stage on a selection of salient tokens, and subsequently executes periodic lightweight updates during decoding. This ensures the low-rank basis continuously adapts to the evolving context with negligible overhead. Our framework is fully compatible with modern attention modules such as FlashAttention (Dao et al., 2022), ensuring practicality in real-world long-context inference.

We evaluate OjaKV on multiple-choice benchmarks from the lm-eval-harness (Biderman et al., 2024), aligning with prior studies (Saxena et al., 2024; Lin et al., 2024). Additionally, for the first time to the best of our knowledge, we evaluate projection-based low-rank KV cache compression methods' performance on generation-centric long-context tasks using LongBench (Bai et al., 2023) and RULER (Hsieh et al., 2024). We also evaluate our method on long reasoning and generation task, AIME (Balunović et al., 2025). Across all settings, OjaKV demonstrates superior performance over static low-rank baselines at the equivalent compression ratios.

In summary, our contributions are threefold. First, we introduce a hybrid low-rank KV-cache compression framework, OjaKV, which combines a selective full-rank storage policy with a context-aware online subspace adaptation. Second, our design is compatible with FlashAttention, ensuring practicality for modern long-context inference pipelines. Third, we conduct the first comprehensive evaluation of low-rank KV compression on challenging generation-centric benchmarks, moving beyond the simpler multiple-choice tasks.

2 Related Work

Recent work on reducing the memory footprint of LLM inference has led to various KV cache compression strategies, including quantization, token pruning, offloading, and subspace compression. Our method builds on low-rank approximation, extending it with an online, context-adaptive formulation. We review relevant methods below.

2.1 KV-Cache Compression

The memory overhead of the KV cache motivates four main compression strategies (Shi et al., 2024). (1) *Quantization*: These methods reduce precision (e.g., to 4-bit or 2-bit) for storage. Approaches like KVQuant (Hooper et al., 2024) and KIVI (Liu et al., 2024c) achieve significant compression with minimal quality loss. (2) *Token selection*: This category retains only essential tokens. StreamingLLM (Xiao et al., 2023) keeps "attention sinks," while SnapKV (Li et al., 2024) selects tokens based on importance scores. (3) *Offloading*: Methods such as Quest (Tang et al., 2024) and ShadowKV (Sun et al., 2024) store the cache in CPU memory, selectively reloading relevant parts

during computation. (4) *Low-rank approximation*: These approaches project keys and values into a lower-dimensional subspace (Saxena et al., 2024; Lin et al., 2024). Our method falls into this last category and is *orthogonal* to the others, enabling additive memory savings. Appendix A.8 provides analysis and experiments showing that our method can be combined with token-eviction-based methods.

2.2 Low-Rank Approximation for Attention

Low-rank structures are widely utilized for compression. Palu (Chang et al., 2024) and ReCalKV (Yan et al., 2025) factorize weights to cache compressed states, though often with accuracy degradation. EigenAttention (Saxena et al., 2024) and MatryoshkaKV (Lin et al., 2024) project activations using bases derived from calibration data ($\boldsymbol{U_k}, \boldsymbol{U_v}$). However, these methods rely on a *static* basis, which degrades performance under distribution shifts. While architectures like Multi-Head Latent Attention (Liu et al., 2024a,b) integrate low-rank structures natively, they require training new models. Our method addresses these limitations via *online subspace adaptation*, continuously updating projection matrices during inference for a plug-in, context-aware approximation.

2.3 Online Principal Component Analysis

In autoregressive decoding, rerunning SVD on the entire history is computationally infeasible. Online PCA algorithms address this by incrementally updating the subspace. Perturbation techniques (Gu and Eisenstat, 1994) provide accuracy but are too resource-intensive for high-dimensional streams. Incremental SVD (Brand, 2002) offers a middle ground but remains costly for real-time LLM inference. In contrast, stochastic methods like Oja's rule (Oja, 1997) efficiently optimize variance via gradient updates. Our proposed method, OjaKV, builds on this foundation. By applying Oja's rule to update key and value subspaces on-the-fly, it enables fast, adaptive, and calibration-free approximation.

3 Preliminaries: Low-Rank Attention

We begin by describing the standard attention mechanism and how it can be adapted for low-rank KV cache compression. For simplicity, we focus on a single attention head. Let the head dimension be d_h and the input sequence be $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. Standard attention first projects the input into query, key, and value representations using projection matrices $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{d \times d_h}$:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q, \qquad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k, \qquad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v,$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d_h}$. The KV cache stores $\boldsymbol{K}$ and $\boldsymbol{V}$ for future use during decoding. The attention scores $\boldsymbol{A}$ are then computed as the scaled dot product between queries and keys, followed by a softmax operation. The output of the attention head, $\boldsymbol{O}$, is computed by applying these attention scores to value matrix $\boldsymbol{V}$: $\boldsymbol{A} = \text{softmax}\left(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d_h}\right) \in^{n \times n}, \boldsymbol{O} = \boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{n \times d_h}$.

3.1 Low-Rank KV-Cache Approximation

The core idea of low-rank approximation is to project $\boldsymbol{K}$ and $\boldsymbol{V}$ onto lower-dimensional subspaces. We define two orthonormal bases: $\boldsymbol{U}_k \in \mathbb{R}^{d_h \times r_k}$ for keys and queries, and $\boldsymbol{U}_v \in \mathbb{R}^{d_h \times r_v}$ for values, where $r_k, r_v \ll d_h$ are the desired ranks for compression. The bases satisfy $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_{r_k}$ and $\boldsymbol{U}_v^\top \boldsymbol{U}_v = \boldsymbol{I}_{r_v}$. The low-rank bases $\boldsymbol{U}_k$ and $\boldsymbol{U}_v$ are initialized from a small calibration dataset (see Appendix A.2 for details).

Instead of caching the full-rank $\boldsymbol{K}$ and $\boldsymbol{V}$, we store their *compressed* representations:

$$\tilde{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{U}_k \in \mathbb{R}^{n \times r_k}, \qquad \tilde{\boldsymbol{V}} = \boldsymbol{V}\boldsymbol{U}_v \in \mathbb{R}^{n \times r_v}.$$

This reduces per-token storage requirement for the KV cache from $2d_h$ to $r_k + r_v$. If needed, the full-rank matrices can be approximately reconstructed via $\hat{\boldsymbol{K}} = \tilde{\boldsymbol{K}}\boldsymbol{U}_k^\top$ and $\hat{\boldsymbol{V}} = \tilde{\boldsymbol{V}}\boldsymbol{U}_v^\top$.

3.2 Efficient Attention Computation

This compressed representation allows for a more efficient attention calculation. Projecting queries into the shared query-key subspace yields $\tilde{\boldsymbol{Q}} = \boldsymbol{Q}\boldsymbol{U}_k$. The attention scores can be computed directly in the low-rank space, as the projection is mathematically equivalent to using the reconstructed matrices:

$$\begin{aligned} \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{K}}^\top &= (\boldsymbol{Q}\boldsymbol{U}_k)(\boldsymbol{K}\boldsymbol{U}_k)^\top \\ &= \boldsymbol{Q}\boldsymbol{U}_k\boldsymbol{U}_k^\top\boldsymbol{K}^\top = \boldsymbol{Q}\boldsymbol{U}_k(\boldsymbol{K}\boldsymbol{U}_k)^\top \\ &= \boldsymbol{Q}\boldsymbol{U}_k\tilde{\boldsymbol{K}}^\top \\ &= \boldsymbol{Q}\hat{\boldsymbol{K}}^\top \in \mathbb{R}^{n \times n} \end{aligned} \tag{1}$$

Thus, attention computed in the low-rank space is equivalent to using the reconstructed keys. The full attention operation is then performed in the low-rank space, and the final output

is projected back to the original dimension: $\tilde{\boldsymbol{O}} = \mathrm{softmax}\left(\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{K}}^{\top}/\sqrt{d_h}\right)\tilde{\boldsymbol{V}} \in \mathbb{R}^{n\times r_v}, \hat{\boldsymbol{O}} = \tilde{\boldsymbol{O}}\boldsymbol{U}_v^{\top} \in \mathbb{R}^{n\times d_h}$.

We note that the final output computation in the above formulation is mathematically equivalent to using the reconstructed keys and values, i.e., $\hat{\boldsymbol{O}} = \mathrm{softmax}\left(\boldsymbol{Q}\hat{\boldsymbol{K}}^{\top}/\sqrt{d_h}\right)\hat{\boldsymbol{V}}$.

3.3 Practical Implementation: Compatibility with FlashAttention

Optimized attention kernels like FlashAttention operate on full-dimensional tensors of shape $(n \times d_h)$ and cannot directly use the compressed features. To maintain compatibility, we store the compressed KV cache ($\tilde{\boldsymbol{K}}$, $\tilde{\boldsymbol{V}}$) and perform on-the-fly reconstruction before using FlashAttention. Given queries $\boldsymbol{Q} \in \mathbb{R}^{n\times d_h}$, we reconstruct the keys and values in the original space from the compressed cache:

$$\hat{\boldsymbol{K}} = \tilde{\boldsymbol{K}}\boldsymbol{U}_k^{\top} \in \mathbb{R}^{n\times d_h}, \qquad \hat{\boldsymbol{V}} = \tilde{\boldsymbol{V}}\boldsymbol{U}_v^{\top} \in \mathbb{R}^{n\times d_h}.$$

These reconstructed tensors are then passed to FlashAttention:

$$\boldsymbol{O}_{\text{out}} = \hat{\boldsymbol{O}} = \mathrm{FlashAttention}(\boldsymbol{Q}, \hat{\boldsymbol{K}}, \hat{\boldsymbol{V}}).$$

This approach maintains the memory savings of a compressed cache while incurring only a modest runtime overhead, as detailed in Appendix A.3.

4 Motivation

Offline low-rank bases are fitted to the calibration distribution and can misalign under domain or task shifts at inference, increasing projection error for keys and values. We therefore maintain an adaptive basis that periodically refreshing the basis with key and value features from current prompts and generated content the model can track distributional shifts and maintain alignment between the low-rank subspace and the evolving sequence. To validate this hypothesis, we conduct an experiment, summarized in Table 1. We compute an initial basis, $\boldsymbol{U}_{\text{cal}}$, from a general-domain corpus (WikiText-2 (Guo et al., 2020)) and evaluate it on a long-context news summarization task from a different domain (the MultiNews subset of LongBench (Bai et al., 2023)). We compare this against an adapted basis, $\boldsymbol{U}_{\text{adapt}}$, formed by updating $\boldsymbol{U}_{\text{cal}}$ online with Oja's rule after processing a short prefix of the MultiNews data. An oracle basis, $\boldsymbol{U}_{\text{test}}$, computed via PCA on the full test set, serves as an upper bound on performance.

Table 1: RER and SO on the MultiNews test set. Adapting the basis online reduces the projection error (RER) and improves alignment with the oracle test basis ($\boldsymbol{U}_{\text{test}}$).

Basis	RER (on Test) ↓	SO with $\boldsymbol{U}_{\text{test}}$ ↑
$\boldsymbol{U}_{\text{cal}}$	0.255	0.597
$\boldsymbol{U}_{\text{adapt}}$	0.097	0.653

We use two metrics: **Residual-Energy Ratio (RER)** (Najafzadeh and Mahmoudi-Rad, 2024) measures the projection error, and **Subspace Overlap (SO)** (Knyazev and Zhu, 2012), which we compute against the oracle basis ($\boldsymbol{U}_{\text{test}}$) to quantify alignment, defined as $\mathrm{SO}(\boldsymbol{U}_1, \boldsymbol{U}_2) = \mathrm{Tr}(\boldsymbol{U}_1^{\top}\boldsymbol{U}_2\boldsymbol{U}_2^{\top}\boldsymbol{U}_1)/r$. The in-domain RER of $\boldsymbol{U}_{\text{cal}}$ on the calibration set is 0.035. As shown in Table 1, the static calibration basis generalizes poorly under distribution shift: its RER increases to 0.255 on the new task. Applying a lightweight Oja update to form $\boldsymbol{U}_{\text{adapt}}$ mitigates this, lowering the RER to 0.097. This adaptation also improves alignment with the oracle basis, raising the SO from 0.597 to 0.653. These findings confirm that online updates can effectively counteract distribution shift.

5 Methodology

Our method, OjaKV, introduces a hybrid strategy for memory-efficient inference that combines selective full-rank retention, with the goal of preserving high-fidelity representations for critical tokens, with online-adapted low-rank compression for remaining sequence. As illustrated in Figure 1, most key and value vectors are projected into a compact subspace via learned projection matrices, which are continuously adapted during inference to remain aligned with the evolving context.

Our framework is built around three core components: (i) a hybrid KV cache storage policy that exempts key contextual tokens from compression; (ii) a lightweight initialization procedure that seeds the projection matrices from a small calibration corpus (see Appendix A.2 for details), (iii) a two-phase online update scheme using Oja's algorithm to adapt the low-rank subspace during both the prompt (prefill) and decoding stages (Appendix A.1).

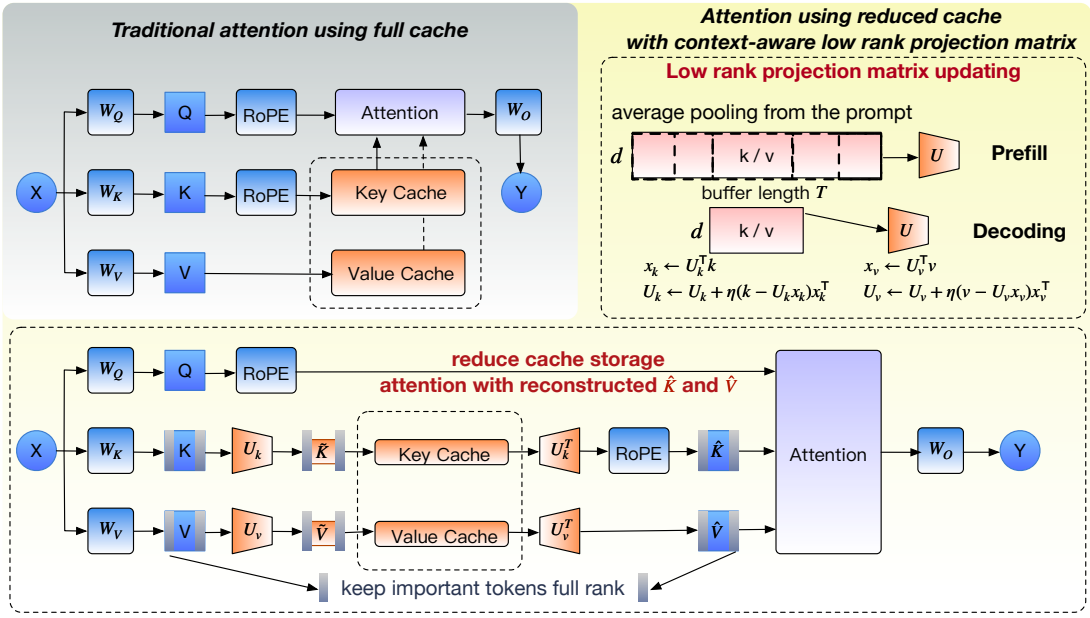

Figure 1: Overview of the OjaKV workflow. The **top-left panel** shows standard attention using full-rank KV caching. Our method, shown in the **bottom panel**, introduces a low-rank path where keys and values are compressed using projection matrices $(\boldsymbol{U}_k, \boldsymbol{U}_v)$ before caching. The **top-right inset** illustrates the core mechanism: these projection matrices are dynamically updated during both the prefill and decoding phases to adapt to the context.

5.1 Hybrid KV Storage with Reconstruction-Error Selection

In long-context generation, not all tokens contribute equally to downstream predictions, and crucially, not all tokens suffer equally from low-rank approximation error. Our hybrid storage policy dynamically identifies which tokens should retain full-rank representations based on their *reconstruction error* under the current low-rank basis.

Error-Based Token Selection. For each key token, we compute the residual between the full-rank key and its low-rank reconstruction:

$$\boldsymbol{r}_t = \boldsymbol{k}_t - \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{k}_t$$

This residual directly governs the attention score perturbation: for any query $\boldsymbol{q}$, the dot-product error introduced by compression, $\Delta = \boldsymbol{q}^\top \boldsymbol{r}_t / \sqrt{d}$, is bounded by $|\Delta| \leq \|\boldsymbol{q}\| \|\boldsymbol{r}_t\| / \sqrt{d}$ via Cauchy-Schwarz. Unlike positional heuristics (e.g., retaining initial tokens as "attention sinks"), which assume reconstruction error is negligible beyond a fixed window, our approach explicitly bounds error by selecting tokens based on their actual residual magnitude.

To assess downstream impact, we weight residuals by attention from recent queries. Let $\boldsymbol{Q}_{\text{win}}$ denote queries from a sliding window of the last w positions. We compute query-weighted error scores:

$$e_t = \frac{1}{|\mathcal{W}_t|} \sum_{h,q \in \mathcal{W}_t} \left| \frac{\boldsymbol{q}_{h,q}^\top \boldsymbol{r}_t}{\sqrt{d_h}} \right|$$

where $\mathcal{W}_t$ denotes query positions that can attend to token t under causal masking. After optional pooling, we retain the top-k tokens with highest error scores at full rank while compressing all other tokens. This ensures that tokens poorly represented by the current basis retain their full expressiveness, while the attention perturbation for all compressed tokens remains bounded with theoretical guarantees that static positional policies lack.

5.2 Two-Phase Online Updates with Oja's Algorithm

The low-rank bases $\boldsymbol{U}_k$ and $\boldsymbol{U}_v$ are initialized to capture the principal subspaces of key and value representations. Recall that in standard attention, the output for query $\boldsymbol{q}_t$ is computed as:

$$\text{Attn}(\boldsymbol{q}_t, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i \leq t} \frac{\exp(\boldsymbol{q}_t^\top \boldsymbol{k}_i / \sqrt{d})}{\sum_{j \leq t} \exp(\boldsymbol{q}_t^\top \boldsymbol{k}_j / \sqrt{d})} \boldsymbol{v}_i$$

The fidelity of our low-rank approximation depends on how well $\boldsymbol{U}_k$ and $\boldsymbol{U}_v$ span the subspaces occupied by keys and values, respectively. Formally, the optimal rank-r basis minimizes the expected

reconstruction error $\mathbb{E}[\|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}\|^2]$, which is achieved by the top-r eigenvectors of the covariance matrix $\boldsymbol{C} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$.

However, during autoregressive generation, newly decoded tokens may introduce key-value patterns that deviate from the initial prompt distribution. A static basis trained only on the prompt will accumulate approximation error as the context grows, degrading attention accuracy. To address this, we update the projection bases online using Oja's rule (Oja, 1982), a streaming algorithm that incrementally tracks the principal subspace of non-stationary data.

Oja's Rule for Subspace Tracking. Given a new sample $\boldsymbol{x}$, Oja's rule updates the basis $\boldsymbol{U} \in \mathbb{R}^{d \times r}$ as:

$$\boldsymbol{U} \leftarrow \boldsymbol{U} + \eta \left(\boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{U} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{U} \right)$$

This can be interpreted as stochastic gradient ascent on $\mathrm{tr}(\boldsymbol{U}^\top \boldsymbol{C}\boldsymbol{U})$ with an implicit orthonormality constraint. The first term $\boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{U}$ pulls the basis toward directions of high variance, while the second term $\boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{U}$ enforces approximate orthogonality. For batch updates with samples $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, we use the empirical covariance $\boldsymbol{C} = \boldsymbol{X}^\top \boldsymbol{X}/n$.

Prefill Stage. During prefill, we initialize the bases using the full set of key and value vectors from the prompt. To reduce redundancy from positionally adjacent tokens and lower computational cost, we apply local average pooling before the update:

$$\boldsymbol{X}_{\text{pooled}} = \mathrm{AvgPool}(\boldsymbol{X}, b),$$
$$\boldsymbol{U} \leftarrow \boldsymbol{U} + \eta \left(\boldsymbol{C}_{\text{pooled}} \boldsymbol{U} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{C}_{\text{pooled}} \boldsymbol{U} \right)$$

where $\boldsymbol{C}_{\text{pooled}} = \boldsymbol{X}_{\text{pooled}}^\top \boldsymbol{X}_{\text{pooled}}/N$, and b is the pooling window size. After the update, we re-orthonormalize via QR decomposition to ensure numerical stability.

Decoding Stage. During decoding, each generated token produces new key-value pairs that may lie outside the current subspace. We accumulate these vectors in a buffer $\mathcal{B}$. Every T steps, we apply Oja's rule to the buffered features:

$$\boldsymbol{U} \leftarrow \boldsymbol{U} + \eta \left(\boldsymbol{C}_{\mathcal{B}} \boldsymbol{U} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{C}_{\mathcal{B}} \boldsymbol{U} \right)$$

with a conservative learning rate to balance adaptation against stability. After the update, we re-orthonormalize the bases and clear the buffer. This periodic refinement ensures that the low-rank subspace continuously tracks the evolving key-value distribution throughout generation.

6 Experiments

In this section, we present the experimental setup and evaluate **OjaKV** against relevant baselines in realistic long-context inference scenarios. All experiments are conducted on NVIDIA H100 GPUs using **Llama-2-7B** and **Llama-3.1-8B** (Grattafiori et al., 2024), evaluated on three diverse benchmarks: **RULER** (Hsieh et al., 2024), **LongBench** (Bai et al., 2023), and **AIME 2025** (Balunović et al., 2025). Additional results on **lm-eval-harness** (Agarwal et al., 2024) are provided in the Appendix A.5.

We compare **OjaKV** against four baselines: (1) **Full KV Cache**, the uncompressed baseline serving as a performance upper bound; (2) **Eigen-N**, a direct low-rank implementation of prior work (Saxena et al., 2024) that is impractical for long contexts due to its incompatibility with FlashAttention; (3) **StaticPCA**, which uses the same fixed, offline-computed SVD basis as Eigen-N but reconstructs full-rank tensors on-the-fly for compatibility with modern inference; and (4) **Palu** (Chang et al., 2024), which decomposes model weights into low-rank matrices. We report model accuracy, memory usage (GB), and Time to First Token (TTFT), with detailed configurations in Appendices A.4 and A.7.

6.1 RULER

Setup. We benchmark OjaKV on the RULER long-context retrieval suite using LongChat-7b-v1.5-32k. We evaluate performance on challenging input sequences of 16K tokens, creating significant GPU memory pressure. We report results under three cache budgets: uncompressed (Full), 20% savings (0.8×), and 40% savings (0.6×). For each compressed budget, we compare StaticPCA, Palu, and Eigen-N baselines against our context-aware OjaKV.

Results. As shown in Table 2, the limitations of existing low-rank methods become apparent on the demanding RULER benchmark. The Eigen-N baseline is infeasible in this setting, as its incompatibility with FlashAttention leads to OOM errors at this sequence length. Palu fails entirely, producing zero accuracy across all tasks. StaticPCA, while functional, suffers substantial degradation—averaging

only 28.37 at $0.8\times$ and 23.44 at $0.6\times$ compression.

In contrast, OjaKV achieves strong retrieval accuracy across all RULER subtasks. These results validate the effectiveness of our dynamic, context-adaptive framework.

Table 2: Retrieval accuracy (%) on the RULER using LongChat-7b-v1.5-32k.

Method	prompt length 16K							
	S1	S2	MK1	MK2	MQ	MV	QA	Avg
FullKV	100	99	91	74	70	71	13	74
Eigen-N 0.8x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	N/A
StaticPCA 0.8x	68	17	23	4	47.75	25.75	13.08	28.37
Palu 0.8x	0	0	0	0	0	0	0	0
OjaKV 0.8x	90	50	40	13	55.75	44.75	20.42	44.85
Eigen-N 0.6x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	N/A
StaticPCA 0.6x	59	5	20	3	30.75	31	15.33	23.44
Palu 0.6x	0	0	0	0	0	0	0	0
OjaKV 0.6x	94	51	37	8	32.25	34.25	23.42	39.99

6.2 LongBench

We further evaluate OjaKV on LongBench, a benchmark designed to test long-context inference across diverse tasks, including single and multi-document QA, few-shot learning, and code generation. The task input lengths vary from a few thousand to over 20K tokens. As shown in Table 3, OjaKV outperforms StaticPCA across both models and compression ratios on the majority of tasks. The performance advantage is less pronounced compared to RULER, potentially because LongBench tasks test for comprehension of a long, stable context. A case study demonstrating how OjaKV improves output quality is provided in Appendix A.6.

6.3 Reasoning Task

In this section we present the experiment setup and evaluate OjaKV using reasoning models. Specifically, we use the distilled reasoning models from R1-Distill, Deepseek-R1-Distill-Llama3-8b (DeepSeek-AI et al., 2025) as the base models. We evaluate OjaKV and baseline methods on math and reasoning task AIME2025.

Reasoning tasks present a unique challenge for KV cache compression: the input prompt is typically short (containing only the problem statement), while the model generates extremely long chains of thought during decoding. In this setting, all methods retain full-rank representations for the prompt tokens, but must compress the vast majority of tokens generated during decoding. This creates a stringent test for low-rank approximation quality, as the reasoning process continuously introduces new semantic patterns that a static basis cannot capture.

As shown in Table 4, static compression methods (Eigen-N, StaticPCA, and Palu) completely fail on AIME2025, achieving 0% accuracy. In contrast, OjaKV maintains 13.0% accuracy by continuously adapting its low-rank bases via online Oja updates throughout decoding. While there remains a gap to the vanilla baseline (43.3%), OjaKV is the only compression method that preserves meaningful reasoning capability, demonstrating the critical importance of online subspace adaptation for long-form generation tasks.

6.4 Efficiency

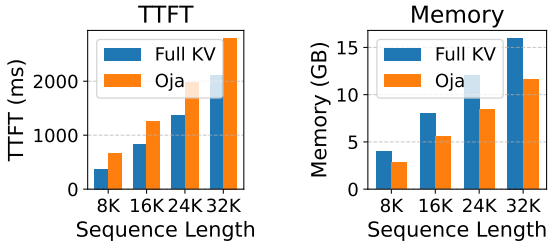

Figure 2: Efficiency comparison of Full KV and OjaKV (60% compression).

We compare OjaKV against the Full KV cache on Llama-3.1-8B-Instruct in terms of latency and GPU memory (Figure 2). The Oja update introduces some overhead to TTFT. At 32K tokens, TTFT increases from 2102 ms (Full KV) to 2801 ms (OjaKV). In contrast, memory usage, which is the limiting factor for long-context inference, decreases from 16 GB to 11.6 GB at 32K tokens. This memory reduction enables longer inputs under the same budget.

Table 5 shows that OjaKV achieves online adaptation with modest latency overhead (10.9–13.4%), while StaticPCA remains nearly as fast as FullKV. Notably, if one were to periodically recompute the PCA basis from scratch (as in StaticPCA) every T tokens, the overhead would be substantially higher than OjaKV's incremental updates. The update interval T thus provides a practical knob to balance basis freshness and computational efficiency.

7 Ablation Studies

To investigate the contribution of each component in OjaKV, we conduct a progressive ablation study

Table 3: Accuracy (%) on LongBench tasks for Llama2-7B and Llama-3.1-8B.

	LLMs	Single-Document QA		Multi-Document QA			Few-shot Learning		Synthetic		Code		Avg
		NrtvQA	MF-en	HotpotQA	2WikiMQA	Musique	TREC	SAMSum	PCount	PRe	Lcc	RB-P	Avg
Llama-2-7B	Full KV	18.79	34.41	25.3	28.33	8.52	0.0	6.22	1.55	9.0	15.08	17.35	14.96
	Eigen-N 0.8x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	NA
	StaticPCA 0.8x	16.95	32.8	21.31	24.73	6.28	0	5.34	2.25	2.61	14.06	16.78	13.01
	Palu 0.8x	15.75	21.32	8.5	10.11	4.34	0	4.14	2.35	6.58	6.48	5.31	7.72
	OjaKV 0.8x	16.28	31.19	20.4	23.7	7.44	0	3.91	1.61	4	16.16	20.85	13.23
	Eigen-N 0.6x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	NA
	StaticPCA 0.8x	14.21	27.54	17.41	19.77	5.11	0.25	5.65	3.01	0.5	13.41	18.84	11.43
	Palu 0.6x	17.17	23.63	9.71	10.22	5.3	0	3.86	3.16	7.1	5.08	4.77	8.18
	OjaKV 0.6x	15.91	31.48	19.97	20.34	7.69	0	5.76	2.01	2.28	16.06	21.15	12.97
Llama-3.1-8B	Full KV	29.56	53.0	53.76	46.13	28.38	7.5	7.47	6.25	99.5	19.88	19.22	33.7
	Eigen-N 0.8x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	NA
	StaticPCA 0.8x	11.81	49.68	48.06	43.56	15.43	10	9.46	3	89.5	21.36	22.46	29.48
	Palu 0.8x	30.54	32.91	21.38	18.07	14.76	71	4.07	3	93.14	5.14	5.86	27.26
	OjaKV 0.8x	12.34	48.89	49.26	44.69	17.47	9	9.59	3	89	21.71	22.92	29.81
	Eigen-N 0.6x	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	NA
	StaticPCA 0.6x	8.16	39.38	25.63	22.73	12.65	0.67	7.1	1.62	12.5	18.76	23.48	15.70
	Palu 0.6x	24.25	28.35	18.72	16.53	10.86	65	4.28	3	20.49	2.81	3.23	17.96
	OjaKV 0.6x	9.73	43.94	35.95	34.49	15.21	3.5	7.78	3.5	33	18.21	21.94	20.66

Table 4: Performance comparison of reasoning models across AIME2025 benchmark. Accuracy is reported in percent (%).

Model	Approach	AIME2025
Llama	Vanilla	43.3
	Eigen-N	0
	StaticPCA	0
	Palu	0
	OjaKV	**13.0**

Table 5: Decoding latency comparison. T denotes the Oja update interval (in tokens) during the decoding phase.

Method	**Latency (ms/token)**
Full KV	36.48
Static PCA	36.60
OjaKV ($T = 64$)	41.38
OjaKV ($T = 128$)	41.09
OjaKV ($T = 256$)	40.44

on Llama-2-7B at 0.6x compression ratio. The results are summarized in Table 6.

Hybrid storage contributes the largest gain, improving average accuracy from 38.9% to 57%. This confirms that retaining high-error tokens at full rank is essential for preserving information poorly captured by the low-rank basis. Online Oja updates provide an additional improvement, demonstrating that adaptive subspace tracking offers complementary benefits.

8 Conclusion

In this work, we addressed the critical KV cache memory bottleneck in long-context LLMs. We introduced **OjaKV**, a novel framework that integrates a **hybrid storage policy** preserving critical tokens in full rank, and a lightweight, Oja-based **online update scheme** to adapt the low-rank subspace for remaining tokens.

Table 6: Ablation study on **RULER** benchmark. We report effective context length and accuracy (%) on representative tasks.

Configuration	**S1**	**S2**	**MK1**	**MQ**	**Avg**
StaticPCA (Baseline)	68	17	23	47.7	38.9
+ Hybrid Storage	88	47	39	54.5	57
+ Hybrid & Online (**OjaKV**)	**90**	**50**	**40**	**55.8**	**60**

Our extensive experiments show that OjaKV consistently outperforms strong static baselines at aggressive compression ratios. Crucially, our evaluation is one of the first to comprehensively assess low-rank methods on challenging, **generation-based** long-context tasks, revealing that while naive uniform compression can harm generation quality, OjaKV's hybrid policy effectively mitigates this by strategically preserving key tokens. OjaKV demonstrates the largest gains on very challenging long-context benchmarks, confirming the value of online subspace adaptation. With full compatibility with FlashAttention and multiplicative savings when combined with token-eviction methods, OjaKV establishes a practical, parameter-free paradigm for efficient long-context LLM inference.

Future Work A promising direction is replacing fixed hyperparameters with dynamic schedules, adapting learning rates and buffer size based on metrics like activation shift.

Limitations

Our method has several limitations. First, OjaKV introduces fixed hyperparameters (learning rates, buffer size, top-k) that may require tuning for different models or tasks. Second, while the Oja update is lightweight, it still incurs additional computational overhead compared to static compression methods. Third, our hybrid selection strategy relies on reconstruction error, which may not perfectly capture token importance for all downstream tasks.

References

Anisha Agarwal, Aaron Chan, Shubham Chandel, Jinu Jang, Shaun Miller, Roshanak Zilouchian Moghaddam, Yevhen Mohylevskyy, Neel Sundaresan, and Michele Tufano. 2024. Copilot evaluation harness: Evaluating llm-guided software programming. *Preprint*, arXiv:2402.14261.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, and 1 others. 2024. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*.

Matthew Brand. 2002. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, pages 707–720. Springer.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. 2024. Palu: Compressing kv-cache with low-rank projection. *Preprint*, arXiv:2407.21118.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Preprint*, arXiv:2205.14135.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

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.

Ming Gu and Stanley C Eisenstat. 1994. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM journal on Matrix Analysis and Applications*, 15(4):1266–1276.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. 2020. Wiki-40b: Multilingual language model dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2440–2452.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Andrew V Knyazev and Peizhen Zhu. 2012. Principal angles between subspaces and their tangents. *arXiv preprint arXiv:1209.0523*.

Gleb Kumichev, Pavel Blinov, Yulia Kuzkina, Vasily Goncharov, Galina Zubkova, Nikolai Zenovkin, Aleksei Goncharov, and Andrey Savchenko. 2024. Medsyn: Llm-based synthetic medical text generation framework. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 215–230. Springer.

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

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, Tianqi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. 2024. Matryoshkakv: Adaptive kv compression via trainable orthogonal projection. *arXiv preprint arXiv:2410.14731*.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024a. Deepseek-v2: A strong, economical, and efficient

mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024b. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Mohammad Najafzadeh and Mohammad Mahmoudi-Rad. 2024. Residual energy evaluation in vortex structures: On the application of machine learning models. *Results in Engineering*, 23:102792.

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Erkki Oja. 1982. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273.

Erkki Oja. 1997. The nonlinear pca learning rule in independent component analysis. *Neurocomputing*, 17(1):25–45.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen attention: Attention in low-rank space for kv cache compression. *arXiv preprint arXiv:2408.05646*.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*.

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

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*.

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

Xianglong Yan, Zhiteng Li, Tianao Zhang, Linghe Kong, Yulun Zhang, and Xiaokang Yang. 2025. Recalkv: Low-rank kv cache compression via head reordering and offline calibration. *arXiv preprint arXiv:2505.24357*.

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

Yuxuan Zhu, Ali Falahati, David H Yang, and Mohammad Mohammadi Amiri. 2025. Sentencekv: Efficient llm inference via sentence-level semantic kv caching. *arXiv preprint arXiv:2504.00970*.

A Appendix

A.1 Oja's Updating Algorithm

We consolidate the complete OjaKV online updating process into a formal procedure in Algorithm 1. The algorithm outlines the two primary stages of our method: a comprehensive update during the prefill phase and lightweight, periodic updates during the decoding phase.

The **Prefilling Phase** is designed to adapt the initial, general-purpose projection matrices (U_k and U_v) to the specific content of the input prompt. First, average pooling with size p is applied to reduce the number of vectors for efficient computation (Line 2). The pooled key and value vectors are then used to perform a batch update on the projection matrices via Oja's rule (Lines 3-4), after which the matrices are re-orthonormalized to maintain their geometric properties (Line 5). Finally, our hybrid storage policy is enacted through an error-weighted selection mechanism: we compute the reconstruction error $\boldsymbol{K} - \hat{\boldsymbol{K}}$ weighted by query attention from a local window, normalize the scores, and select the top-k tokens with the highest error to be stored at full rank (Lines 6-7).

The **Decoding Phase** handles the continuous adaptation of the subspace as new tokens are autoregressively generated. At each step, the newly generated key-value pair is temporarily stored in a

buffer (Line 11). To maintain efficiency, updates are performed periodically. Every T steps, average pooling is applied to the buffered vectors, followed by another batch Oja update with a more conservative learning rate to ensure stable learning (Lines 13-14). The bases are again re-orthonormalized, and the buffers are cleared for the next cycle (Lines 15-16). This two-phase approach allows OjaKV to make a strong initial adaptation to the context while continuously refining the subspace with minimal overhead during generation.

Algorithm 1 OjaKV

Require: Low rank projection matrices $\boldsymbol{U}_k, \boldsymbol{U}_v$; learning rates η; update buffer size T; pool size p; top-k for selection

1: **Prefilling Phase:**
2: Form matrices $\boldsymbol{K}, \boldsymbol{V}$; apply average pooling with size p
3: $\tilde{\boldsymbol{K}} \leftarrow \boldsymbol{U}_k^\top \boldsymbol{K}$; $\boldsymbol{U}_k \leftarrow \boldsymbol{U}_k + \eta\big(\boldsymbol{K} - \boldsymbol{U}_k \tilde{\boldsymbol{K}}\big)\tilde{\boldsymbol{K}}^\top$
4: $\tilde{\boldsymbol{V}} \leftarrow \boldsymbol{U}_v^\top \boldsymbol{V}$; $\boldsymbol{U}_v \leftarrow \boldsymbol{U}_v + \eta\big(\boldsymbol{V} - \boldsymbol{U}_v \tilde{\boldsymbol{V}}\big)\tilde{\boldsymbol{V}}^\top$
5: $(\boldsymbol{U}_k, \boldsymbol{U}_v) \leftarrow \text{Orthonormalise}(\boldsymbol{U}_k, \boldsymbol{U}_v)$
6: $\hat{\boldsymbol{K}} \leftarrow \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{K}$; $\boldsymbol{E} \leftarrow |\boldsymbol{Q}_{\text{window}}^\top (\boldsymbol{K} - \hat{\boldsymbol{K}})|$ ▷ Error-weighted selection
7: $\mathcal{I}_{\text{full}} \leftarrow \text{Top-}k(\text{Normalize}(\boldsymbol{E}))$
8:
9: **Decoding Phase:**
10: **for** step $t = 1, 2, \ldots$ **do**
11: Generate new (k_t, v_t) and append to buffers $\mathcal{B}_k, \mathcal{B}_v$
12: **if** $t \bmod T = 0$ **then**
13: $\tilde{\boldsymbol{K}} \leftarrow \boldsymbol{U}_{\boldsymbol{k}}^\top \boldsymbol{K}$; $\boldsymbol{U}_{\boldsymbol{k}} \leftarrow \boldsymbol{U}_{\boldsymbol{k}} + \eta\big(\boldsymbol{K} - \boldsymbol{U}_{\boldsymbol{k}} \tilde{\boldsymbol{K}}\big)\tilde{\boldsymbol{K}}^\top$
14: $\tilde{\boldsymbol{V}} \leftarrow \boldsymbol{U}_{\boldsymbol{v}}^\top \boldsymbol{V}$; $\boldsymbol{U}_v \leftarrow \boldsymbol{U}_v + \eta\big(\boldsymbol{V} - \boldsymbol{U}_v \tilde{\boldsymbol{V}}\big)\tilde{\boldsymbol{V}}^\top$
15: $(\boldsymbol{U}_k, \boldsymbol{U}_v) \leftarrow \text{Ortho}(\boldsymbol{U}_k, \boldsymbol{U}_v)$
16: Reset $\mathcal{B}_k, \mathcal{B}_v$
17: **end if**
18: **end for**

A.2 Low rank subspace initialization

We describe here the detailed procedure for constructing the initial projection bases.

For attention head i, we gather per-token activations of queries, keys, and values from n_s sampled sequences of length n:

$$\mathbf{R}_i^Q = \big[(\mathbf{Q}_i^1)^\top, \ldots, (\mathbf{Q}_i^{n_s})^\top\big]$$
$$\mathbf{R}_i^K = \big[(\mathbf{K}_i^1)^\top, \ldots, (\mathbf{K}_i^{n_s})^\top\big]$$
$$\mathbf{R}_i^V = \big[(\mathbf{V}_i^1)^\top, \ldots, (\mathbf{V}_i^{n_s})^\top\big]$$

where each $\mathbf{R}_i^{(\cdot)} \in \mathbb{R}^{(n_s \cdot n) \times d_h}$ and d_h is the head dimension.

To encourage a shared representation, we concatenate the query and key matrices:

$$\mathbf{R}_i^{KQ} = [\,\mathbf{R}_i^Q, \mathbf{R}_i^K\,] \in \mathbb{R}^{(n_s \cdot n) \times 2d_h}.$$

Applying compact SVD gives

$$\mathbf{R}_i^{KQ} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\top,$$

with singular values $\sigma_1 \geq \cdots \geq \sigma_{d_h}$.

We select the smallest rank r satisfying the energy criterion

$$\frac{\|(\mathbf{R}_i^{KQ})_r\|_F^2}{\|\mathbf{R}_i^{KQ}\|_F^2} \;\geq\; \epsilon_{\text{th}}.$$

The top-r columns of $\boldsymbol{U}$ define the query–key basis $\boldsymbol{U}_{\boldsymbol{k}} \in \mathbb{R}^{d_h \times r_k}$.

For the values, we apply the same procedure directly to $\mathbf{R}_i^V$ to obtain $\boldsymbol{U}_{\boldsymbol{v}} \in \mathbb{R}^{d_h \times r_v}$. Finally, to maintain consistency across attention heads in a layer, we set the effective rank to the maximum r observed in that layer.

A.3 Equivalence to Full-Rank FlashAttention and Cost Comparison

Notation Let $\boldsymbol{Q} \in \mathbb{R}^{m \times d_h}$ and $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d_h}$ denote the per-head query, key, and value blocks, where m is the number of current queries and n is the number of cached keys/values. At the prefilling stage, $m = n$; at the decoding stage, $m = 1$. Let $\boldsymbol{U}_k \in \mathbb{R}^{d_h \times r_k}$ and $\boldsymbol{U}_v \in \mathbb{R}^{d_h \times r_v}$ be orthonormal bases with $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_{r_k}$ and $\boldsymbol{U}_v^\top \boldsymbol{U}_v = \boldsymbol{I}_{r_v}$. Define compressed features

$$\tilde{\boldsymbol{Q}} = \boldsymbol{Q}\boldsymbol{U}_k \in \mathbb{R}^{m \times r_k}$$
$$\tilde{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{U}_k \in \mathbb{R}^{n \times r_k}$$
$$\tilde{\boldsymbol{V}} = \boldsymbol{V}\boldsymbol{U}_v \in \mathbb{R}^{n \times r_v}$$

A.3.1 Equivalence of two computation regimes

We compare (a) computing attention in the reduced space and expanding the output, versus (b) reconstructing full-rank $\boldsymbol{K}, \boldsymbol{V}$ and calling a standard FlashAttention kernel.

Low-rank kernel (compute-then-expand). Form reduced logits and outputs

$$\tilde{\boldsymbol{A}} \;=\; \text{softmax}\Big(\frac{\tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big) \in \mathbb{R}^{m \times n}$$
$$\tilde{\boldsymbol{O}} \;=\; \tilde{\boldsymbol{A}}\,\tilde{\boldsymbol{V}} \in \mathbb{R}^{m \times r_v},$$

then expand $\hat{\boldsymbol{O}} = \tilde{\boldsymbol{O}}\,\boldsymbol{U}_v^\top \in \mathbb{R}^{m \times d_h}$.

FlashAttention-compatible (reconstruct-then-compute). Reconstruct full-rank tensors

$$\hat{\boldsymbol{K}} = \tilde{\boldsymbol{K}}\,\boldsymbol{U}_k^\top \in \mathbb{R}^{n\times d_h}, \qquad \hat{\boldsymbol{V}} = \tilde{\boldsymbol{V}}\,\boldsymbol{U}_v^\top \in \mathbb{R}^{n\times d_h},$$

and call FlashAttention with the original queries $\boldsymbol{Q}$:

$$\hat{\boldsymbol{O}} \;=\; \mathrm{softmax}\Big(\frac{\boldsymbol{Q}\,\hat{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big)\,\hat{\boldsymbol{V}} \;\in\; \mathbb{R}^{m\times d_h}.$$

Lemma (logit equivalence). With the above definitions,

$$\boldsymbol{Q}\,\hat{\boldsymbol{K}}^\top \;=\; \tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top.$$

Proof. Since $\hat{\boldsymbol{K}}^\top = (\tilde{\boldsymbol{K}}\,\boldsymbol{U}_k^\top)^\top = \boldsymbol{U}_k\,\tilde{\boldsymbol{K}}^\top$, we have $\boldsymbol{Q}\,\hat{\boldsymbol{K}}^\top = \boldsymbol{Q}\,(\boldsymbol{U}_k\,\tilde{\boldsymbol{K}}^\top) = (\boldsymbol{Q}\,\boldsymbol{U}_k)\,\tilde{\boldsymbol{K}}^\top = \tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top$.

Corollary (output equivalence). The two computation regimes produce the same output $\hat{\boldsymbol{O}}$. *Proof.* Starting from the FlashAttention-compatible definition of $\hat{\boldsymbol{O}}$:

$$\begin{aligned}
\hat{\boldsymbol{O}} &= \mathrm{softmax}\Big(\frac{\boldsymbol{Q}\,\hat{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big)\,\hat{\boldsymbol{V}} && \text{(FA-compatible)}\\
&= \mathrm{softmax}\Big(\frac{\tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big)\,\hat{\boldsymbol{V}} && \text{(logit equivalence)}\\
&= \mathrm{softmax}\Big(\frac{\tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big)(\tilde{\boldsymbol{V}}\boldsymbol{U}_v^\top) && \text{(substitute } \hat{\boldsymbol{V}}\text{)}\\
&= \Big(\mathrm{softmax}\Big(\frac{\tilde{\boldsymbol{Q}}\,\tilde{\boldsymbol{K}}^\top}{\sqrt{d_h}}\Big)\tilde{\boldsymbol{V}}\Big)\boldsymbol{U}_v^\top && \text{(associativity)}\\
&= \tilde{\boldsymbol{O}}\,\boldsymbol{U}_v^\top && \text{(low-rank kernel)}
\end{aligned}$$

Hence the FlashAttention-compatible path is numerically equivalent to computing in the reduced space and then expanding, provided the same scaling $1/\sqrt{d_h}$ is used. Using $1/\sqrt{r_k}$ changes the effective temperature and usually needs calibration.

A.3.2 Complexity and memory comparison

We summarize per-head costs for a single block with m queries against n cached keys/values. Big-O ignores softmax and masking; General matrix multiply (GEMM) shapes are shown for clarity.

Discussion. The low-rank kernel reduces the quadratic dot-product costs from $O(mnd_h)$ to $O(mnr_k)$ and $O(mnr_v)$, plus a linear expansion cost of $O(mr_vd_h)$. The FA-compatible path keeps the full-rank kernel complexity $O(mnd_h)$ but preserves memory savings by storing only $\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}$; the reconstruction GEMMs are linear in n.

KV-cache memory in bytes. Let b be bytes per scalar (e.g., b=2 for float16). For L layers and H_{kv} KV heads, the total KV memory for a sequence of length T and batch size B is

$$\begin{aligned}
\text{Full rank:}\quad M_{\text{full}} &= B\,T\,L\,H_{\text{kv}}\,(2d_h)\,b,\\
\text{Low rank:}\quad M_{\text{low}} &= B\,T\,L\,H_{\text{kv}}\,(r_k + r_v)\,b,
\end{aligned}$$

with fractional saving

$$\text{Saving} \;=\; 1 - \frac{r_k + r_v}{2d_h}.$$

When r_k=r_v=r, this simplifies to Saving $= 1 - \frac{r}{d_h}$.

A.4 Detailed Experimental Setup

This section provides a detailed overview of the experimental environment, models, datasets, and evaluation protocols used in this study to ensure full reproducibility of our results.

Hardware and Software Environment. All experiments were conducted on a single **NVIDIA H100 NVL** GPU. The software stack was built upon PyTorch and Hugging Face Transformers. The specific versions of the core libraries were as follows: **PyTorch** `torch==2.6.0`, **Transformers** `transformers==4.44.0`, and **FlashAttention** `flash_attn==2.7.4.post1`. All models were run using their standard float16 precision implementation.

Models. We evaluated our method on several prominent open-source Large Language Models. For clarity and reproducibility, the specific Hugging Face repository identifiers for each model were: **Llama-2-7B** (`meta-llama/Llama-2-7b-chat-hf`), **Llama-3.1-8B** (`meta-llama/Llama-3.1-8B-Instruct`), and **LongChat-7B for RULER** (`lmsys/longchat-7b-v1.5-32k`).

Calibration Dataset. The initial low-rank projection bases, U_k and U_v, were derived from a small, general-domain calibration dataset. For this purpose, we used the **WikiText-2** dataset. The initialization process followed the procedure outlined in Appendix A.2, where key and value activations were collected from a number of sampled sequences and then decomposed via SVD to form the initial subspaces.

Evaluation Benchmarks and Metrics. Our comprehensive evaluation was performed across three diverse benchmarks: **lm-eval-harness**, **LongBench**, and **RULER**. Performance was assessed

Table 7: Computational complexity comparison.

Regime	Main computations	KV memory
Full-rank baseline	$\boldsymbol{Q}\boldsymbol{K}^\top$: $O(mnd_h)$; $\boldsymbol{A}\boldsymbol{V}$: $O(mnd_h)$	$2d_h$
Low-rank kernel	$\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{K}}^\top$: $O(mnr_k)$; $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{V}}$: $O(mnr_v)$ Expand: $O(mr_vd_h)$	$r_k + r_v$
FA-compatible	Reconstruct $\boldsymbol{K}, \boldsymbol{V}$: $O(n(r_k + r_v)d_h)$; FA: $O(mnd_h)$	$r_k + r_v$

based on the following metrics. **Accuracy**: We report the specific accuracy metrics as defined by each benchmark's protocol. For lm-eval-harness, this includes the zero-shot accuracy on tasks like PiQA, WinoGrande, and HellaSwag. For LongBench and RULER, this corresponds to their respective scoring mechanisms for long-context reasoning and retrieval tasks. **GPU Memory**: Memory consumption is reported in Gigabytes (GB) and reflects the specific GPU memory allocated to KV cache during the inference process for a given context length. This provides a practical measure of the hardware requirements. **Latency**: Latency is reported as Time To First Token (TTFT) in milliseconds (ms), which primarily measures the overhead during the prompt processing (prefill) stage. This is a critical metric for user-facing applications where initial response time is important.

A.5 Lm-eval-harness

To gauge the impact of KV cache compression on downstream utility, we follow the lm-eval-harness protocol on five diverse zero-shot benchmarks: *PiQA*, *WinoGrande*, *ARC-Easy*, *ARC-Challenge*, and *HellaSwag*. We evaluate all methods on the relatively short-context tasks within the lm-eval-harness, where sequence lengths are typically limited to a few hundred tokens. As shown in Table 8, we make two key observations in this setting. First, the Eigen-N and StaticPCA baselines yield identical results. This finding empirically validates our analysis in Appendix A.3, confirming the numerical equivalence between the native low-rank kernel and our FlashAttention-compatible implementation. Second, while OjaKV achieves accuracy very close to the full-rank baseline, StaticPCA-H also performs very well, suggesting that the impact of keeping a full rank attention sink and recent window, has a significant impact for these short context-tasks, as these key values contribute to a larger percentage of the total KV cache. Shorter contexts are also more robust to compression, as there is minimal accuracy drop at $0.6\times$ compression for both Llama-2-7B and Llama-3.1-8B.

Overall, our experiments on these three different benchmarks demonstrate the versatility of OjaKV, and shows that it can perform best in scenarios with long, dynamic context like in RULER.

A.6 Qualitative Analysis and Case Studies

To complement the quantitative results presented in the main paper, this section provides a qualitative case study. Our goal is to illustrate the practical impact of OjaKV's online subspace adaptation on generation quality, particularly in long-context scenarios where the distribution of activations can shift significantly. We use a long-document summarization task from the MultiNews subset of the LongBench benchmark, comparing summaries generated by our full OjaKV framework against the StaticPCA baseline at an aggressive 0.6x compression ratio.

> **Input Document Snippet (Conceptual Summary)**
>
> The input is a lengthy (over. 8k tokens) article. It is a multi-doc summarization that requires summarizing over multiple news.

> **Reference Summary**

Table 8: Accuracy (%) on lm-eval-harness tasks for Llama-2-7B and Llama-3.1-8B.

Model	Compression Ratio	Method	Acc ↑					
			WinoG	PiQA	HellaS	Arc-e	Arc-c	Avg-Acc
Llama-2-7B	Full	Baseline	66.38	76.39	57.80	73.86	44.20	63.73
	0.8x	Eigen-N	65.90	75.03	56.29	71.30	41.38	61.98
		StaticPCA	65.90	75.03	56.29	71.30	41.38	61.98
		OjaKV	**66.30**	75.79	57.51	**73.86**	**44.37**	63.57
	0.6x	Eigen-N	62.98	74.54	54.99	70.03	39.93	60.49
		StaticPCA	62.98	74.54	54.99	70.03	39.93	60.49
		OjaKV	**66.30**	**76.39**	**57.06**	**73.74**	**44.37**	**63.57**
Llama-3.1-8B	Full	Baseline	73.88	80.03	59.04	81.86	51.88	69.34
	0.8x	Eigen-N	73.56	79.65	57.74	81.65	51.54	68.83
		StaticPCA	73.56	79.65	57.74	81.65	51.54	68.83
		OjaKV	**73.95**	**79.92**	**59.14**	**81.90**	**51.79**	**69.34**
	0.6x	Eigen-N	69.85	78.67	54.87	79.38	48.12	66.18
		StaticPCA	69.85	78.67	54.87	79.38	48.12	66.18
		OjaKV	**73.95**	**79.82**	**58.35**	**81.90**	**51.88**	**69.18**

The Occupy LA encampment got a stay of execution this morning—though a few arrests have been reported beyond the boundaries of City Hall Park. Two hours after protesters defied city hall's order to vacate the park by 12:01am, lawyer and Occupy supporter Jim Lafferty told Reuters he expected police to give the camp a two-day reprieve. At around 5am, police told protesters over a loudspeaker that they didn't intend to clear the park, the LA Times reports. Police do, however, intend to clear protesters out of the street, and several people who refused to move out of an intersection have already been arrested. Ẅe definitely need to get morning traffic in,ä police commander explained. Everything also seems calm at Occupy Philadelphia, where protesters defied a 5pm Sunday deadline to disperse. But as of last night, a police inspector said they didn't intend to forcibly evict anyone. Ẅe look forward to working with Occupy Philadelphia and a resolution of the problem,ḧe told the Philadelphia Inquirer. Confrontation is never good.

Baseline: StaticPCA (0.6x Compression)

The Occupy Wall Street movement has been making headlines in various cities across the United States. In Philadelphia, Occupy Philly protesters remained on site at City Hall into the evening of November 28, despite a deadline set by Mayor Nutter for them to leave. The protesters, who had set up camp on the City Hall lawn, were given a deadline to vacate the area, but they refused to leave. The police had no plans to evict anyone, and the protesters were allowed to stay on the lawn.

Analysis of Baseline: The summary generated by StaticPCA exhibits a classic failure mode in long-context summarization: **topical fixation and critical information loss.** The source document clearly discusses events in two distinct locations, with a primary focus on the tense standoff in Los Angeles and a secondary mention of the calm situation in Philadelphia. However, the StaticPCA summary focuses *exclusively* on Philadelphia, completely omitting any mention of the events in Los Angeles, which constituted the main subject of the report. This suggests that the static principal components, established early, failed to represent the new, dominant information introduced later. The model's inability to adapt to this semantic shift resulted in a factually incomplete and highly misleading summary.

OjaKV (0.6x Compression)

> The Occupy Wall Street movement continues to gain momentum across the United States, with protests taking place in various cities. In Los Angeles, the Occupy LA encampment at City Hall Park was the site of a tense standoff between protesters and police, with commanders later announcing they would not clear the encampment. In Philadelphia, the Occupy Philadelphia campers also remained in place despite a 5 p.m. deadline to vacate the area. The movement is a national one protesting economic inequality, and the response from law enforcement has varied by city.

Analysis of OjaKV: In stark contrast, the summary from OjaKV successfully captures the multifaceted nature of the source document. It correctly identifies and synthesizes the key events from **both Los Angeles and Philadelphia**, presenting a coherent and comprehensive overview. This demonstrates the effectiveness of OjaKV's online subspace adaptation. As the model processed the document and encountered new information related to the LA protest, it dynamically updated its KV cache's principal components. This adaptation allowed it to preserve the crucial details from different sections of the long-context input, avoiding the catastrophic information loss seen in the StaticPCA baseline. The resulting summary is significantly more accurate and useful.

A.7 Default hyperparameters

Table 9: Default hyperparameters unless stated otherwise.

Symbol	Default	Description
η	0.10	Oja update lr
T	32	decode update period (steps)
n	–	prompt length
w	32	importance window size (queries)

A.8 Compatibility with Sequence Length Compression

Our OjaKV method compresses the *feature dimension* ($d \to r$) of the key and value vectors. As a result, it is orthogonal and compatible with sequence length compression techniques the *sequence length* ($n \to m$) such as token eviction or selection. This compatibility allows their benefits to be compounded for multiplicative savings. We briefly analyze this property theoretically here and validate it experimentally in Appendix A.8.

A token eviction policy can be represented by a selector matrix $\mathbf{S} \in^{n \times m}$. For a fixed, linear selector, our low-rank projection $\mathbf{U}^\top$ associates perfectly with the selection operation:

$$\mathbf{U}_k^\top(\mathbf{K}\mathbf{S}) = (\mathbf{U}_k^\top\mathbf{K})\mathbf{S}, \quad \mathbf{U}_v^\top(\mathbf{V}\mathbf{S}) = (\mathbf{U}_v^\top\mathbf{V})\mathbf{S}.$$

For advanced, context-dependent selectors where $\mathbf{S}_t = \mathrm{Sel}(\mathbf{K}, \mathbf{Q})$ is a function that selects the most relevant tokens based on the current query $\mathbf{Q}$, the commutation is not exact, but the additional projection error is bounded by the error of the selection policy itself:

$$\begin{aligned}\left\|\boldsymbol{U}^\mathsf{T}\boldsymbol{K} - \boldsymbol{U}^\mathsf{T}\boldsymbol{K}\mathbf{S}_t\right\|_F &\leq \left\|\boldsymbol{K} - \boldsymbol{K}\mathbf{S}_t\right\|_F \\ &= \left\|\mathbf{K} - \mathrm{Sel}(\mathbf{K}, \mathbf{Q})\right\|_F.\end{aligned} \tag{2}$$

because left-multiplication by $\boldsymbol{U}^\mathsf{T}$ is a contraction with respect to the Frobenius norm. This operational orthogonality means that combining a rank-r OjaKV with a policy retaining m of n tokens yields a total compression ratio of $\mathrm{CR} = (d/r) \times (n/m)$.

We also provides empirical validation for this claim by combining OjaKV with SnapKV (Li et al., 2024), a representative token selection method.

Experimental Setup. We chose SnapKV as it is a strong baseline that uses importance scores to identify and retain salient tokens. We evaluated four configurations on the LongBench benchmark suite using the Llama-3.1-8B model. The configurations are: (1) the uncompressed baseline, (2) SnapKV alone with a 50% token keep rate, (3) OjaKV alone with a 0.6x rank compression, and (4) a combined approach applying both OjaKV's rank compression and SnapKV's token eviction. Performance is measured by the average accuracy across LongBench tasks, and efficiency is measured by the total KV cache compression ratio.

Results and Analysis. The results, presented in Table 10, confirm our hypothesis. Our analysis shows that **OjaKV** can be effectively combined with token eviction methods like SnapKV. This compounded approach further reduces KV cache memory usage with only a minor, graceful degradation in model accuracy. This result validates that our feature-dimension compression is complementary to sequence-length compression, offering a practical path to even greater memory efficiency.

Table 10: Compounded KV cache compression by combining OjaKV with SnapKV. The total compression ratio demonstrates multiplicative savings, offering a compelling trade-off between performance and memory efficiency.

Method	**Rank Comp.**	**Token Keep Rate**	**Memory Usage (%)**	**Accuracy**
Full KV Cache (Baseline)	1.0x	100%	100%	53.0
SnapKV (Token Sel. only)	1.0x	50%	50%	52.66
OjaKV (Rank Comp. only)	1.67x (0.6x)	100%	60%	43.13
OjaKV + SnapKV	**1.67x (0.6x)**	**50%**	**30%**	**43.33**