KVCAPSULE: EFFICIENT SEQUENTIAL KV COMPRESSION FOR VLMS WITH ASYMMETRIC REDUNDANCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Visual tokens can dominate the key-value (KV) cache of long-context vision-language models (VLMs), increasing persistent memory and decoding traffic. Many visual sequence-reduction methods prune or merge positions using importance estimates obtained before or early in decoding, making discarded KV states unavailable if their relevance changes later. We show that visual KV exhibits sequence-level redundancy, decoding-dependent attention, and distinct redundancy structures for keys and values. Based on these observations, we introduce KVCapsule, an asymmetric visual-KV compression framework that reduces persistent cache storage while maintaining an approximate full visual-position representation for subsequent attention. KVCapsule reconstructs visual keys from selectively retained positions and compresses visual values using sequence-level PCA, while keeping the pretrained VLM backbone frozen. Across VLM backbones and image and video benchmarks, KVCapsule achieves a strong quality-memory trade-off. At 10K context and batch size 1, it reduces the measured persistent KV-related footprint by 55.0% and improves decoding throughput by 24.0%; at larger-batch long-context settings, throughput improves by up to 2×, while our analytical memory model predicts up to a 2.4× smaller persistent footprint beyond the break-even regime.

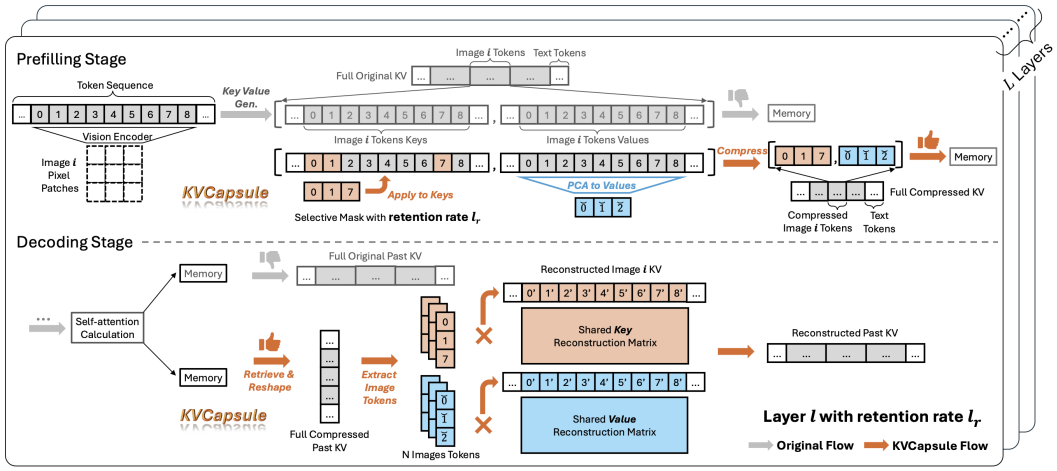


Figure 1: **Overview of the KVCapsule framework.** The gray region denotes the original process of VLMs, while the yellow region shows the process of KVCapsule.

1 INTRODUCTION

Extending LLMs to the multimodal domain has led to rapid progress in VLMs, enabling applications such as image captioning, visual question answering, and multimodal dialogue. Recent open-source state-of-the-art VLMs, including Qwen-VL Bai et al. (2023), LLaVA-NeXT Liu et al. (2024a), and InternVL Zhu et al. (2025), demonstrate increasingly strong multimodal reasoning capabilities.

Despite this progress, VLMs face a major scalability bottleneck during autoregressive decoding: the memory required to store and move the KV cache. This issue is particularly severe for vision tokens,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

which are encoded at higher granularity than text Li et al. (2025); Huang et al. (2025). A single image is typically represented by hundreds or thousands of tokens, so even a few images can substantially increase the effective context length and rapidly exhaust GPU memory.

This challenge is particularly severe in multi-image and long-context settings, including visual comparison, reasoning over image sequences, video understanding, and multi-turn multimodal dialogue Alayrac et al. (2022); Zhang et al. (2023); Suhr et al. (2019). In these settings, visual tokens can dominate the input context and remain in the KV cache throughout decoding, causing memory usage to grow with both visual-context length and generated sequence length. Efficient KV-cache reduction is therefore critical for scalable VLM inference under limited memory budgets.

To mitigate KV cache growth, prior work has explored compression techniques including quantization Hooper et al. (2024); Cheng et al. (2025), low-rank approximation Zhang et al. (2024a); Chang et al. (2025b), and attention-based pruning Li et al. (2024); Wan et al. (2024), with extensions to VLMs Su et al. (2025); Han et al. (2025); Saxena et al. (2024); Mu et al. (2025); Tu et al. (2024); Chang et al. (2025a); Pei et al. (2024); Jiang et al. (2025); Yang et al. (2025); Huang et al. (2025). These techniques target complementary dimensions of KV storage. In particular, many visual-token pruning and merging methods reduce sequence length using importance estimates obtained during prefill or early decoding. Once a visual position is discarded, its per-position KV state is no longer available to later queries. However, visual attention can vary across layers, heads, and decoding steps, in long-context and multi-image settings. A position that receives little attention early may become important later, making static position-reduction decisions less reliable throughout decoding.

Our analysis shows that visual-token importance changes substantially across layers, heads, and decoding steps. This creates a fundamental tension for sequence-reduction methods: visual KV states are highly redundant and therefore compressible, but the positions that can be safely removed are not necessarily static throughout generation.

Rather than asking which visual positions can be permanently discarded, we ask whether the visual sequence can be stored compactly while keeping the original position space approximately addressable during decoding. To this end, we propose KVCapsule, a backbone-frozen KV cache compression framework for vision tokens. Rather than permanently removing low-ranked tokens, KVCapsule stores sequence-level representations of the visual KV cache before attention score computation, as shown in Figure 1. Importantly, KVCapsule uses asymmetric compression for keys and values: it reconstructs keys to preserve attention geometry, while compressing values along the sequence dimension to retain their dominant semantic subspace. At decoding time, the compressed states recover an approximate full-position visual representation before attention, allowing subsequent decoding queries to attend over the reconstructed visual positions. This design separates memory reduction from irreversible token removal.

KVCapsule does not fine-tune the pretrained VLM backbone. It requires only one-time offline calibration of the key-reconstruction, selection, and value-projection operators, which are fixed during inference. Evaluated across multiple VLM architectures and benchmark datasets, our method achieves substantial memory savings with strong quality–memory trade-off. We summarize the key contributions as follows:

- **Sequential redundancy discovery.** We identify a high degree of sequential redundancy in visual KV cache (Section 2.1).
- **Attention dynamics analysis.** We show that vision-token attention varies across layers, heads, and decoding steps, challenging static importance assumptions (Section 2.2).
- **Asymmetric KV redundancy.** We empirically demonstrate a disparity between visual keys and values, which motivates our asymmetric strategy for KV cache compression (Section 2.3).
- **Recoverable sequential KV compression.** We compress visual KV along the sequence dimension while reconstructing full-position representations for attention, preserving access to positions that may become important later (Section 3.1).
- **Simple integration.** KVCapsule leaves the pretrained VLM parameters unchanged, adding only lightweight compression and reconstruction components (Section 3.2).
- **Efficient inference.** KVCapsule provides a strong quality-memory trade-off across long-context and multi-image settings, while fused KVCapsule folds reconstruction into attention to reduce KV-memory traffic and improve decoding throughput (Sections 4.1, 4.2, and 4.3).

2 OBSERVATIONS

In this section, we analyze the structure and decoding behavior of visual KV states using Qwen3-VL-8B-Instruct Bai et al. (2023) on 500 randomly sampled examples from MEGABench Chen et al. (2025). We identify three design-driving properties: sequence-level redundancy among vision tokens, dynamic vision token importance during decoding, and distinct structural characteristics of keys and values. These findings expose a mismatch between visual KV behavior and static-importance pruning methods and motivate treating visual keys and values with different compression operators.

2.1 SEQUENTIAL REDUNDANCY

Visual inputs contain spatial correlations and object-level coherence, creating redundancy across vision tokens. This redundancy is already visible during prefill, when the multimodal input is encoded into the KV cache. As shown in Figure 2, attention forms image-level blocks induced by spatial coherence, while the instruction selectively attends to a compact set of salient visual regions. This structured attention suggests that visual positions exhibit substantial correlation and motivates examining whether their KV states contain exploitable sequence-level redundancy.

Figure 2: Prefill attention structure. Attention forms image-level blocks and selective cross-modal patterns over vision tokens.

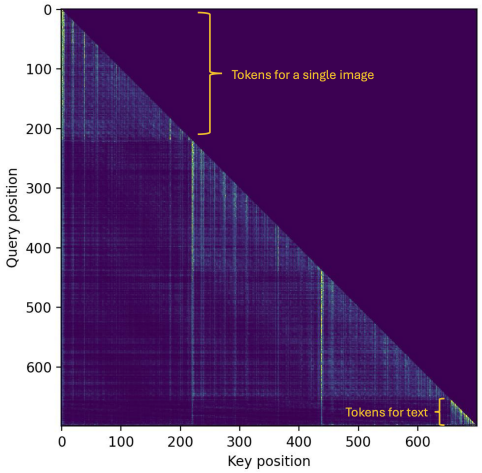


Table 1: Top-10% attended vision-token overlap across decoding steps. For step s, let $\mathcal{T}_s$ denote the $k = \lceil 0.1 N_v \rceil$ most-attended visual positions. Each entry reports $\mathrm{Overlap}(i, j) = |\mathcal{T}_i \cap \mathcal{T}_j|/k$.

Step j	**Reference step** i				
	0	**1**	**5**	**10**	**20**
1	0.42	–	–	–	–
5	0.16	0.33	–	–	–
10	0.16	0.42	0.42	–	–
20	0.25	0.42	0.50	0.54	–
40	0.25	0.33	0.42	0.71	0.66
60	0.25	0.33	0.42	0.58	0.46
Avg. overlap	0.25	0.37	0.44	0.61	0.56

2.2 SEQUENTIAL DYNAMICS OF VISION-TOKEN IMPORTANCE

Structured redundancy does not imply static importance. During autoregressive decoding, the model may require different visual evidence for different generated tokens, making prefill-only or early-decoding importance estimates unreliable. Figure 3 shows that visual attention shifts across decoding steps and layers, with previously low-attention tokens becoming important later. Table 1 quantifies this change: the top-10% token overlap between step 0 and steps 5 or 10 is only 0.16, with an average overlap of 0.25 relative to step 0. Overlap between later decoding steps also remains incomplete, indicating that visual-token importance continues to change throughout generation.

This dynamic behavior exposes a limitation of irreversible position removal: a position discarded using an early importance estimate cannot contribute to later queries if its relevance increases. KVCapsule therefore compresses the visual sequence while retaining a recoverable representation of the original position space. Appendix L directly evaluates whether this design better preserves information associated with positions that become important later in decoding.

2.3 ASYMMETRIC REDUNDANCY IN KV STATES

Beyond decoding dynamics, visual keys and values exhibit distinct sequence-level structures, suggesting that they should not be compressed with the same mechanism. To characterize this asymmetry,

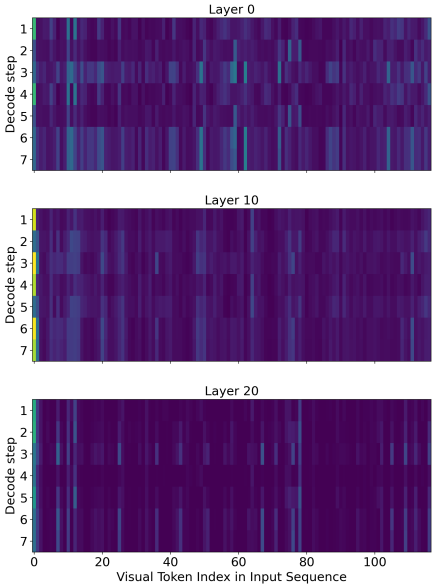


Figure 3: **Vision-token attention dynamics.** Attention shifts across decoding steps and layers; brighter colors indicate higher attention over vision tokens.

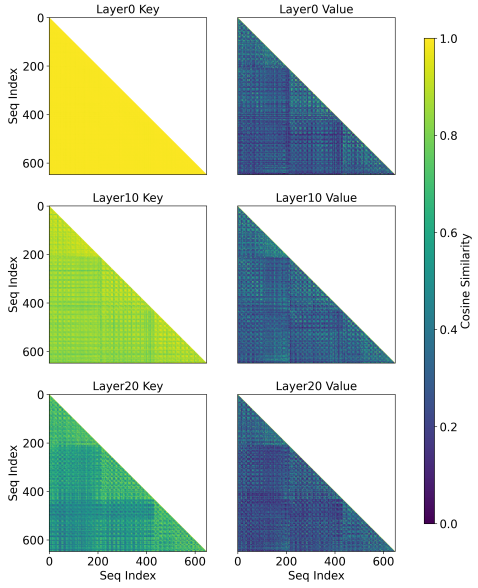


Figure 4: **Key-value similarity across layers.** Keys show stronger early-layer redundancy, while values are more token-specific, motivating asymmetric compression.

we measure pairwise cosine similarity between key and value vectors along the vision-token sequence dimension. As shown in Figure 4, keys have substantially higher inter-token similarity than values, especially in early layers.

The higher inter-token similarity of visual keys suggests stronger redundancy in their query-matching representations. Because keys affect attention through the query-key product, this motivates retaining a compact set of representative keys and learning a reconstruction operator that approximates the original key sequence. However, because vision token importance changes during decoding, permanently pruning the remaining keys can still remove information needed later. We therefore retain representative keys and reconstruct the full key sequence before attention, reducing persistent KV storage while preserving recoverable query-key geometry.

Values exhibit much lower pairwise similarity, indicating that individual value tokens carry more token-specific visual content. Directly pruning value rows can therefore discard localized semantic information. However, low pairwise similarity does not rule out low-rank structure of the full value matrix: pairwise cosine similarity measures local token-to-token closeness, whereas PCA captures dominant global modes of variation across the sequence. This motivates sequence-level PCA for values, which compresses along the token dimension while preserving contributions from all vision tokens. Appendix B further shows that visual values are dense along the hidden dimension, supporting sequence-level compression over hidden-dimension latent compression.

This observation motivates the asymmetric design of KVCapsule: keys are selectively retained and reconstructed to preserve attention geometry, while values are compressed by sequence-level PCA to preserve distributed semantic content without dropping entire tokens. Appendix A quantifies the K/V redundancy gap; Appendix M provides a matched-memory comparison against symmetric K/V designs; and Appendix R evaluates the two branches using output-level fidelity metrics.

2.4 Layer-wise Compression Tolerance

Figure 4 shows that visual KV redundancy is layer-dependent: early-layer keys exhibit stronger global similarity, while deeper-layer keys become more structured and less uniformly redundant; values remain comparatively token-specific across layers. However, high redundancy does not necessarily imply high compression tolerance, since errors in shallow layers can propagate through later layers. We therefore directly measure layer-wise reconstruction sensitivity in Appendix F and use the

resulting trend to define a pyramid retention schedule, with higher retention in compression-sensitive earlier layers and stronger compression in later layers.

3 KVCAPSULE FRAMEWORK

3.1 LEARNING COMPRESSION AND RECONSTRUCTION

The observations above motivate KVCapsule's asymmetric, recoverable design: keys are selectively retained and reconstructed to preserve attention geometry, while values are compressed by sequence-level PCA to preserve distributed semantic content without irreversible token pruning.

In standard multi-head attention, the attention output for queries $\mathbf{Q} \in \mathbb{R}^{s \times d_{\text{head}}}$ is computed as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{\text{head}}}}\right)\mathbf{V}. \tag{1}$$

where $\mathbf{K} \in \mathbb{R}^{m \times d_{\text{head}}}$ and $\mathbf{V} \in \mathbb{R}^{m \times d_{\text{head}}}$ denote the cached keys and values, and m is the sequence length. The query-key product determines attention weights, while the attention-value product aggregates semantic content into the output. This functional distinction motivates different compression strategies for keys and values.

Selective Retention and Reconstruction for Keys. Keys determine attention weights through their similarity to queries. Therefore, key compression should preserve the geometry of the query-key product rather than merely retain tokens with high static attention scores. Motivated by the high inter-token redundancy of visual keys, KVCapsule stores a compact subset of representative keys and reconstructs the full key sequence before attention. This reduces persistent KV storage while avoiding permanent removal of keys that may become important at later decoding steps.

Let $\mathcal{M}_\ell \subseteq 1, \ldots, n$ denote the retained key indices at layer ℓ, with $r_\ell = |\mathcal{M}_\ell| = \lceil \ell_r n \rceil$ and $\tilde{\mathbf{K}}_\ell \in \mathbb{R}^{r_\ell \times d_{\text{head}}}$, where $\ell_r < 1$ is the retention ratio. We reconstruct the full key sequence using a factorized linear operator:

$$\hat{\mathbf{K}}_\ell = \mathbf{W}_{2,\ell}\mathbf{W}_{1,\ell}\tilde{\mathbf{K}}_\ell,$$

where $W_{1,\ell} \in \mathbb{R}^{d_b \times r_\ell}$, $W_{2,\ell} \in \mathbb{R}^{n \times d_b}$, and d_b is the intermediate reconstruction dimension. We set $d_b = n/4$ in all experiments and denote $\theta_\ell = \{W_{1,\ell}, W_{2,\ell}\}$. The reconstruction is factorized through an intermediate bottleneck while remaining fully linear, allowing the reconstruction to be reassociated with query-key computation during fused attention and avoiding explicit materialization of the reconstructed key tensor.

The mask $\mathcal{M}_\ell$ and reconstructor θ_ℓ are jointly calibrated from uncompressed visual-key activations by minimizing reconstruction error together with a regularizer that enforces the target retention ratio. After calibration, both the mask and reconstruction factors are fixed during inference. Appendix C provides the calibration details.

Sequence-Level Low-Rank Compression for Values. Values are directly aggregated into the attention output and carry token-specific visual content. Rather than selecting a subset of value tokens, KVCapsule compresses values along the sequence dimension using PCA.

For each layer ℓ and KV head h, let $V_{\ell,h} \in \mathbb{R}^{n \times d_{\text{head}}}$ denote the visual value matrix. Using the fixed centering vector $\mu_{\ell,h} \in \mathbb{R}^{d_{\text{head}}}$, we center the values as

$$V_{\ell,h}^{\text{center}} = V_{\ell,h} - \mathbf{1}_n \mu_{\ell,h}^\top. \tag{2}$$

PCA is fitted independently for each layer and KV head along the visual sequence dimension using centered calibration activations. Retaining the top $r_\ell = \lceil \ell_r n \rceil$ components gives $U_{\ell,h} \in \mathbb{R}^{r_\ell \times n}$ and

$$\widetilde{V}_{\ell,h} = U_{\ell,h} V_{\ell,h}^{\text{center}}. \tag{3}$$

The approximate full-length value sequence is reconstructed as

$$\widehat{V}_{\ell,h} = U_{\ell,h}^\top \widetilde{V}_{\ell,h} + \mathbf{1}_n \mu_{\ell,h}^\top. \tag{4}$$

The PCA basis is fitted offline, independently for each layer and KV head, from centered calibration activations along the visual sequence dimension. Both the centering vector and PCA basis are fixed after calibration and reused for all subsequent inputs; no per-input fitting is performed at inference. Retaining the leading r_ℓ components gives the optimal rank-r_ℓ approximation of the centered calibration activations under the Frobenius norm, while preserving contributions from all visual positions in the compressed representation. Algorithm 2 and Appendix C specify the calibration-data layout and fitting procedure. Appendices E and O report reconstruction ablations and calibration analysis, respectively.

3.2 Inference

At inference time, all calibrated masks, key reconstructors, PCA bases, and centering vectors are fixed. Following the layer-wise compression tolerance observed in Section 2.4, KVCapsule uses a pyramid retention schedule, retaining more visual KV information in earlier layers and compressing deeper layers more aggressively. The operators are indexed by per-block visual-token length and reused across images or sampled video frames with the same visual-tokenization configuration, so increasing the number of visual blocks does not increase operator size. Appendix Q discusses support for different resolutions, image counts, and video lengths.

Prefill Stage During prefill, KVCapsule initializes a compressed KV cache for vision tokens. At each layer ℓ, visual keys are compressed by the learned mask $\mathcal{M}_\ell$, and visual values are centered and projected onto the sequence-level PCA basis $\mathbf{U}_\ell$. Only the compressed visual keys and values are stored in the persistent cache. Text-token KV states are left unchanged, preserving the original language-cache behavior of the backbone VLM.

Decoding Stage The compressed states define approximate full-position visual K/V representations:

$$\hat{\mathbf{K}}_\ell = \mathbf{W}_{2,\ell}\mathbf{W}_{1,\ell}\tilde{\mathbf{K}}_\ell, \qquad \hat{\mathbf{V}}_\ell = \mathbf{U}_\ell^\top \tilde{\mathbf{V}}_\ell + \mathbf{1}_n \boldsymbol{\mu}_\ell^\top.$$

An explicit reference implementation can materialize these tensors and pass them to standard attention. For efficient inference, however, our fused formulation algebraically reassociates the same linear operators with the attention computation and avoids materializing either full-length tensor (Section 4.3). Both paths produce the same reconstructed attention computation; the fused path changes the execution order rather than the underlying approximation.

4 Experiments

We evaluate KVCapsule for accuracy preservation, robustness across image and video tasks, generalization across VLM backbones, and inference efficiency. The main experiments use approximately 60% visual-KV compression with a pyramid layer-wise schedule. Since visual-token reduction does not translate directly to KV-memory reduction, Appendix K reports measured persistent-KV reduction and throughput. Appendix P compares methods at approximately 40%, 60%, and 70% measured KV-memory reduction. Additional sensitivity analyses appear in Appendices N, F, and I.

4.1 Image-based Evaluation on VLMEvalKit

Table 2 reports VLMEvalKit Duan et al. (2024) results on MME Zhang et al. (2024b), MMMU Yue et al. (2024), COCO Lin et al. (2015), MMBench Liu et al. (2024b), LLaVABench Liu et al. (2023b;a), and HallusionBench Guan et al. (2024), covering perception, reasoning, captioning, instruction following, hallucination robustness, and multi-image understanding. Across five VLM backbones, KVCapsule remains competitive with Full KV and compression baselines, with task-dependent variation across models. Appendix P reports comparisons at nearby persistent-memory operating points, and Appendix M isolates the effect of asymmetric K/V compression.

4.2 Video Understanding Evaluation

To evaluate KVCapsule beyond static images, Table 3 reports results on Video-MME Fu et al. (2025) and MMBench-Video Fang et al. (2024), LongVideoBench Wu et al. (2024), MLVU Zhou

Table 2: Performance comparison of KVCapsule and other baselines on VLMEvalKit across VLMs.

		MME	MMMU	COCO	MMBench	LLaVABench	HallusionBench
Qwen2	Baseline: Full KV	1687.70	0.42	14.28	0.82	80.80	60.12
	KVCapsule	1634.22	**0.42**	14.40	**0.83**	74.80	**60.99**
	FastV	1498.46	0.42	15.00	0.68	55.50	58.68
	SparseVLMs	1637.74	0.41	14.28	0.81	75.30	60.74
	PruMerge	1637.64	0.42	14.28	0.81	74.70	60.74
Qwen2.5	Baseline: Full KV	1685.80	0.20	15.40	0.80	85.90	65.60
	KVCapsule	1635.80	**0.20**	14.40	0.79	**83.70**	**62.20**
	FastV	155.30	0.14	5.90	0.00	12.90	25.60
	SparseVLMs	1637.64	0.20	15.40	0.80	80.20	60.74
	PruMerge	1637.64	0.20	15.40	0.80	78.20	60.74
Qwen3	Baseline: Full KV	1735.60	0.52	15.70	0.80	74.80	53.90
	KVCapsule	**1708.49**	**0.48**	**15.80**	0.78	**73.80**	53.70
	FastV	561.25	0.35	8.01	0.39	22.30	47.83
	SparseVLMs	1706.44	0.34	15.60	0.80	72.00	47.03
	PruMerge	1560.20	0.48	15.70	0.73	73.40	53.80
LLaVA-Mistral	Baseline: Full KV	1458.12	0.40	14.39	0.68	67.80	40.80
	KVCapsule	**1458.10**	**0.41**	14.42	0.67	**67.80**	40.80
	FastV	1440.69	0.37	15.24	0.68	67.70	40.80
	SparseVLMs	1339.93	0.39	15.60	0.64	67.80	45.11
	PruMerge	1457.62	0.39	14.30	0.68	66.50	48.15
LLaVA-Llama	Baseline: Full KV	1515.40	0.46	14.70	0.72	66.30	37.90
	KVCapsule	**1533.40**	**0.60**	**14.70**	**0.71**	**66.00**	49.40
	FastV	1525.65	0.46	14.50	0.69	64.30	43.60
	SparseVLMs	1504.79	0.45	14.40	0.69	65.20	50.00
	PruMerge	1526.40	0.49	14.30	0.71	65.20	52.79

Evaluated models: `Qwen2-VL-7B-Instruct`, `Qwen2.5-VL-7B-Instruct`, `Qwen3-VL-8B-Instruct`, `llava-v1.6-mistral-7b-hf`, `llama3-llava-next-8b-hf`.

et al. (2024), VCRBench Qi et al. (2025), and Video-MMLU Song et al. (2025). Across the three evaluated VLM backbones, KVCapsule remains competitive with Full KV on several long-video and generation-sensitive tasks. Appendix Q further evaluates visual contexts from 8 to 64 frames, while Appendix L measures preservation of information that becomes important later in decoding.

Table 3: Performance comparison of KVCapsule and other baselines on video datasets across VLMs.

		MME(long)	MMBench(fine)	LongVideoBench	MLVU	VCRBench	Video-MMLU CAP	Video-MMLU QA
Qwen2	Baseline: Full KV	0.44	1.23	0.56	0.56	0.25	0.11	0.12
	KVCapsule	**0.44**	1.23	**0.55**	**0.53**	**0.31**	**0.10**	0.09
	FastV	0.42	**1.24**	0.54	0.53	0.28	0.06	**0.11**
	SparseVLMs	0.44	1.22	0.54	0.52	0.31	0.07	0.08
	PruMerge	0.43	1.22	0.54	0.52	0.31	0.09	0.08
Qwen2.5	Baseline: Full KV	0.46	1.22	0.56	0.54	0.23	0.16	0.27
	KVCapsule	**0.44**	**1.01**	**0.55**	**0.50**	**0.22**	**0.17**	**0.38**
	FastV	0.43	0.08	0.33	0.20	0.00	0.03	0.00
	SparseVLMs	0.44	0.73	0.54	0.50	0.20	0.17	0.22
	PruMerge	0.43	0.96	0.54	0.49	0.20	0.08	0.06
Qwen3	Baseline: Full KV	0.50	1.56	0.60	0.58	0.39	0.19	0.39
	KVCapsule	**0.50**	**1.58**	**0.59**	**0.58**	**0.38**	**0.19**	**0.39**
	FastV	0.30	0.59	0.31	0.30	0.19	0.01	0.00
	SparseVLMs	0.49	1.24	0.52	0.50	0.33	0.09	0.32
	PruMerge	0.43	1.53	0.58	0.56	0.37	0.18	0.36

4.3 Efficiency Analysis

Inference Latency. As shown in Algorithm 1, our fused formulation algebraically folds the linear reconstruction operators into decode-time attention, avoiding materialization of the reconstructed visual KV tensors. For values, instead of reconstructing $\hat{\mathbf{V}}_\ell$ before attention, we use associativity:

Algorithm 1 Fused KVCapsule for Layer ℓ

Input: query $\mathbf{Q}$; text KV $(\mathbf{K}_{\text{text}}, \mathbf{V}_{\text{text}})$; compressed visual KV $(\tilde{\mathbf{K}}_\ell, \tilde{\mathbf{V}}_\ell)$; key reconstruction factors $(\mathbf{W}_{1,\ell}, \mathbf{W}_{2,\ell})$; value PCA $(\mathbf{U}_\ell, \boldsymbol{\mu}_\ell)$; scaling σ
Output: Attention output Y
// Step 1: Compute text and compressed visual scores
$L_{\text{text}} \leftarrow \textsc{MatMul}(\mathbf{Q}, \mathbf{K}_{\text{text}}^\top)$
$S_{\text{ret}} \leftarrow \textsc{MatMul}(\mathbf{Q}, \tilde{\mathbf{K}}_\ell^\top)$
// Step 2: Fold key reconstruction into score computation
$S_{\text{mid}} \leftarrow \textsc{MatMul}(S_{\text{ret}}, \mathbf{W}_{1,\ell}^\top)$
$L_{\text{img}} \leftarrow \textsc{MatMul}(S_{\text{mid}}, \mathbf{W}_{2,\ell}^\top)$
$[P_{\text{text}}, P_{\text{img}}] \leftarrow \textsc{Softmax}\left([L_{\text{text}}, L_{\text{img}}] \cdot \sigma\right)$
// Step 3: Fold value reconstruction into attention output
$A_{\text{img}} \leftarrow \textsc{MatMul}(P_{\text{img}}, \mathbf{U}_\ell^\top)$
$Y_{\text{img}} \leftarrow \textsc{MatMul}(A_{\text{img}}, \tilde{\mathbf{V}}_\ell) + (P_{\text{img}}\mathbf{1}_n)\boldsymbol{\mu}_\ell^\top$
$Y_{\text{text}} \leftarrow \textsc{MatMul}(P_{\text{text}}, \mathbf{V}_{\text{text}})$
$Y \leftarrow Y_{\text{text}} + Y_{\text{img}}$
Return Y

$$P_{\text{img}}\left(\mathbf{U}_\ell^\top \tilde{\mathbf{V}}_\ell + \mathbf{1}_n \boldsymbol{\mu}_\ell^\top\right) = (P_{\text{img}}\mathbf{U}_\ell^\top)\tilde{\mathbf{V}}_\ell + (P_{\text{img}}\mathbf{1}_n)\boldsymbol{\mu}_\ell^\top .$$

This projects the full-position visual attention probabilities into the compressed value space, allowing the final multiplication to operate on the compressed value cache without materializing the reconstructed full-length value tensor. For keys, fused KVCapsule similarly computes scores from the compressed keys and applies the reconstruction factors in score space before softmax, avoiding explicit materialization of the full-length key cache. Fusion therefore trades additional score-space computation for lower KV-memory traffic and intermediate materialization.

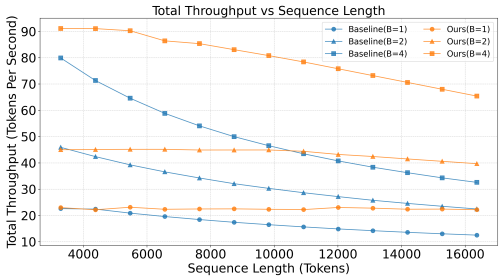
For one decode query, one attention head, and one visual block of length n, the fused key and value paths require $O(r_\ell d_{\text{head}} + r_\ell d_b + n d_b)$ and $O(n r_\ell + r_\ell d_{\text{head}})$ operations, respectively. With $d_b = n/4$, score-space reconstruction contains an $O(n^2)$ term in the per-block visual length. For repeated blocks sharing the tokenization configuration, the calibrated operators are reused as in Appendix Q. Appendix J establishes equivalence between explicit and fused formulations.

Despite the additional reconstruction arithmetic, reduced KV-memory traffic provides a net decoding benefit in the evaluated regime. At 10K context and batch size 1 on one AMD MI250X, the default $c60$ configuration reduces the complete persistent KV-related footprint by 55.0% and increases end-to-end decoding throughput from 16.16 to 20.05 TPS (24.0%). We separately profile attention-path latency across 9K-15K inputs in Figure 5(c), where KVCapsule remains comparatively stable as context length increases.

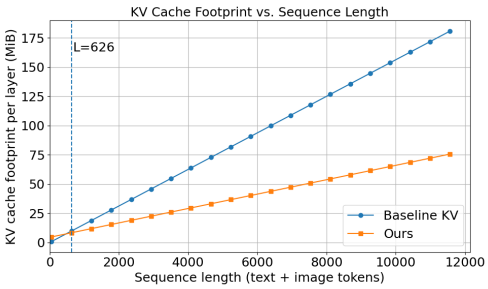
Theoretical Memory Footprint. We decompose persistent memory into request-specific KV storage and fixed model-side operator state. The full-cache baseline stores KV states for $(c + n + t)$ tokens, whereas KVCapsule replaces the n visual states with $r_\ell = \lceil \ell_r n \rceil$ compressed states, reducing the request-specific term to $(c + r_\ell + t)$. The reconstruction factors, PCA bases, centering vectors, and selection metadata are shared across requests with the same visual-tokenization configuration. Because this operator state is fixed while request-specific KV storage grows with context and batch size, the reported configuration reaches a break-even point at $L \approx 626$ tokens, as shown in Figure 5(b). Appendix D provides the complete per-layer memory breakdown.

5 Related Work

Quantization-aware KV cache compression. These methods reduce KV cache memory by lowering tensor precision. Su et al. (2025) propose adaptive quantization for multimodal inputs, while Han et al. (2025) introduce calibration to preserve accuracy after KV quantization. These methods act element-wise and do not reduce sequence length. In contrast, KVCapsule reduces visual sequence storage while preserving numerical precision, making sequence compression complementary to KV quantization.



(a) Throughput scaling on AMD MI250X.



(b) Persistent KV memory footprint.

Method / Attention latency (ms/token)	9K	10K	11K	12K	13K	14K	15K
Full KV	74.58	73.23	76.47	79.69	82.91	86.05	89.16
KVCapsule	**70.22**	**67.02**	**68.46**	**67.69**	**67.00**	**67.58**	**66.18**
Attention-latency reduction	**5.8%**	**8.5%**	**10.5%**	**15.1%**	**19.2%**	**21.5%**	**25.8%**

(c) Attention latency at batch size 1 across 9K-15K input lengths.

Figure 5: **Efficiency analysis of fused KVCapsule.** (a) Fused KVCapsule improves decoding throughput over the full-cache baseline. (b) KVCapsule reduces persistent KV memory after the break-even point at $L \approx 626$ tokens. (c) Attention latency across 9K-15K inputs; Full-KV latency increases with context length, while KVCapsule remains comparatively stable.

Low-rank and subspace approximations. Low-rank methods compress KV states through lower-dimensional projections. Saxena et al. (2024) use eigen decomposition, Zhang et al. (2024a) use orthogonal projections, and Mu et al. (2025) explore layer-wise subspace alignment. In contrast, KVCapsule uses asymmetric sequence-level compression: values are projected along the visual sequence dimension with PCA, while keys retain selected visual positions and reconstruct the full key sequence through a factorized linear operator. This preserves an approximate full visual-position representation for attention from the compressed cache.

Attention-guided token selection and pruning. Prior methods reduce visual KV by retaining positions selected through attention, learned gating, cross-modal similarity, early-exit criteria, top-K relevance, or adaptive spatial selection Tu et al. (2024); Chang et al. (2025a); Pei et al. (2024); Jiang et al. (2025); Yang et al. (2025); Huang et al. (2025). These methods rely on importance estimates from prefill or early decoding, making removed positions unavailable if their relevance changes later. Our analysis shows that visual attention varies across decoding steps and that keys and values exhibit distinct redundancy structures. KVCapsule instead compresses visual KV into a recoverable representation and evaluates attention over an approximate full-position sequence rather than restricting attention to retained positions.

6 Conclusion

We presented KVCapsule, an asymmetric visual-KV compression framework that reconstructs visual keys with a factorized linear operator and compresses values using sequence-level PCA. This design reduces persistent KV storage while preserving an approximate full visual-position representation for attention. Across VLM backbones and image/video benchmarks, KVCapsule provides a strong quality-memory trade-off while improving decoding efficiency. At 10K context and batch size 1, it reduces measured persistent KV-related footprint by 55.0% and improves decoding throughput by 24.0%. At larger-batch long-context settings, throughput improves by up to $2\times$. Our analytical memory model predicts up to a $2.4\times$ smaller persistent footprint beyond the break-even regime.

The fixed reconstruction operators are shared across inputs with the same visual-tokenization configuration, while fused attention avoids materializing the reconstructed full-length K/V tensors. Overall, KVCapsule provides a recoverable sequence-level compression strategy that reduces visual-KV storage without permanently restricting later attention to the retained visual positions.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript editing and improving clarity, as well as to provide feedback on the presentation of the methodology, experiments, and interpretation of results. The authors independently designed the proposed method, conducted the experiments, analyzed the results, and verified all technical claims and AI-assisted revisions.

REPRODUCIBILITY STATEMENT

We provide the algorithmic formulation and inference procedure of KVCapsule in the main text, with additional implementation and calibration details, hyperparameter settings, sensitivity analyses, memory breakdown, and system evaluation provided in the appendices. The evaluated models and benchmarks are identified in the experimental sections, and additional results and ablations are provided in the supplementary material. These materials are intended to provide the information necessary to reproduce the method and reported evaluations.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.

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

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. Palu: Kv-cache compression with low-rank projection. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Jiacheng Chen, Tianhao Liang, Sherman Siu, Zhengqing Wang, Kai Wang, Yubo Wang, Yuansheng Ni, Wang Zhu, Ziyan Jiang, Bohan Lyu, Dongfu Jiang, Xuan He, Yuan Liu, Hexiang Hu, Xiang Yue, and Wenhu Chen. Mega-bench: Scaling multimodal evaluation to over 500 real-world tasks. 2025.

Wen Cheng, Shichen Dong, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2542–2550, 2025.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, et al. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201, 2024.

Xinyu Fang, Kangrui Mao, Haodong Duan, Xiangyu Zhao, Yining Li, Dahua Lin, and Kai Chen. Mmbench-video: A long-form multi-shot benchmark for holistic video understanding. *Advances in Neural Information Processing Systems*, 37:89098–89124, 2024.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. In *CVPR*, 2025.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14375–14385, 2024.

Insu Han, Zeliang Zhang, Zhiyuan Wang, Yifan Zhu, Susan Liang, Jiani Liu, Haiting Lin, Mingjie Zhao, Chenliang Xu, Kun Wan, et al. Calibquant: 1-bit kv cache quantization for multimodal llms. *arXiv preprint arXiv:2502.14882*, 2025.

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

Kai Huang, Hao Zou, Bochen Wang, Ye Xi, Zhen Xie, and Hao Wang. Aircache: Activating inter-modal relevancy kv cache compression for efficient large vision-language model inference. *arXiv preprint arXiv:2503.23956*, 2025.

Zhonghua Jiang, Kunxi Li, Yiyun Zhou, Sihao Liu, Zhaode Wang, Shengyu Zhang, et al. Purekv: Plug-and-play kv cache optimization with spatial-temporal sparse attention for vision-language large models. *arXiv preprint arXiv:2510.25600*, 2025.

Kunxi Li, Zhonghua Jiang, Zhouzhou Shen, ZhaodeWang ZhaodeWang, Chengfei Lv, Shengyu Zhang, Fan Wu, and Fei Wu. Madakv: Adaptive modality-perception kv cache eviction for efficient multimodal long-context inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13306–13318, 2025.

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

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2023a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023b.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024a. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pp. 216–233. Springer, 2024b.

Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. Sals: Sparse attention in latent space for kv cache compression. *arXiv preprint arXiv:2510.24273*, 2025.

Xiaohuan Pei, Tao Huang, and Chang Xu. Cross-self kv cache pruning for efficient vision-language inference. *arXiv preprint arXiv:2412.04652*, 2024.

Yukun Qi, Yiming Zhao, Yu Zeng, Xikun Bao, Wenxuan Huang, Lin Chen, Zehui Chen, Jie Zhao, Zhongang Qi, and Feng Zhao. Vcr-bench: A comprehensive evaluation framework for video chain-of-thought reasoning. *arXiv preprint arXiv:2504.07956*, 2025.

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

Enxin Song, Wenhao Chai, Weili Xu, Jianwen Xie, Yuxuan Liu, and Gaoang Wang. Video-mmlu: A massive multi-discipline lecture understanding benchmark, 2025. URL https://arxiv.org/abs/2504.14693.

Zunhai Su, Wang Shen, Linge Li, Zhe Chen, Hanyu Wei, Huangqi Yu, and Kehong Yuan. Akvq-vl: Attention-aware kv cache adaptive 2-bit quantization for vision-language models. *arXiv preprint arXiv:2501.15021*, 2025.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 6418–6428, 2019.

Dezhan Tu, Danylo Vashchilenko, Yuzhe Lu, and Panpan Xu. Vl-cache: Sparsity and modality-aware kv cache compression for vision-language model inference acceleration. *arXiv preprint arXiv:2410.23317*, 2024.

Zhongwei Wan, Ziang Wu, Che Liu, Jinfa Huang, Zhihong Zhu, Peng Jin, Longyue Wang, and Li Yuan. Look-m: Look-once optimization in kv cache for efficient multimodal long-context inference. *arXiv preprint arXiv:2406.18139*, 2024.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding, 2024. URL https://arxiv.org/abs/2407.15754.

Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Chendi Li, Jinghua Yan, Yu Bai, Ponnuswamy Sadayappan, Xia Hu, et al. Topv: Compatible token pruning with inference time optimization for fast and low-memory multimodal vision language model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19803–19813, 2025.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of CVPR*, 2024.

Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*, 2023.

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

Yi-Fan Zhang, Huanyu Zhang, Haochen Tian, Chaoyou Fu, Shuangqing Zhang, Junfei Wu, Feng Li, Kun Wang, Qingsong Wen, Zhang Zhang, et al. Mme-realworld: Could your multimodal llm challenge high-resolution real-world scenarios that are difficult for humans? *arXiv preprint arXiv:2408.13257*, 2024b.

Junjie Zhou, Yan Shu, Bo Zhao, Boya Wu, Shitao Xiao, Xi Yang, Yongping Xiong, Bo Zhang, Tiejun Huang, and Zheng Liu. Mlvu: A comprehensive benchmark for multi-task long video understanding. *arXiv preprint arXiv:2406.04264*, 2024.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A Quantitative Analysis of Key-Value Asymmetry

To strengthen our observation, we conduct a quantitative analysis on 500 randomly selected MEGABench samples. As shown in Table 4, keys consistently exhibit much higher inter-token cosine similarity than values across representative layers. For example, key similarity remains high from shallow to deeper layers, while value similarity stays substantially lower. This supports our observation that keys contain stronger inter-token redundancy, whereas values are more token-specific and should avoid direct token dropping. These results support our asymmetric design choice of applying structure-preserving reconstruction to keys and PCA-based subspace compression to values.

Table 4: Inter-token cosine similarity of keys and values across representative layers, measured on 500 randomly selected MEGABench samples.

Layer	Key Similarity	Value Similarity
Layer 0	0.99	0.35
Layer 10	0.87	0.40
Layer 20	0.66	0.30

B Hidden-Dimension Compression Is Inefficient for Visual Values

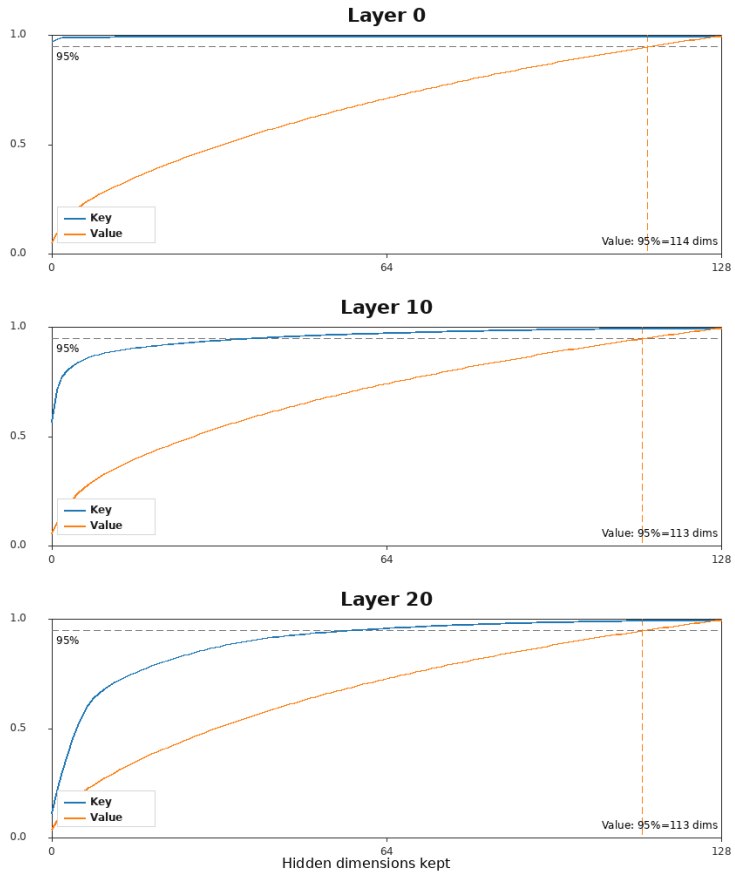


Figure 6: **Hidden-dimension PCA energy for visual KV states.** We compute cumulative PCA energy over feature channels for visual keys and values. Values require nearly the full hidden dimension to preserve 95% variance across layers, suggesting that hidden-dimension compression is inefficient for visual values.

Table 5: Hidden-dimension PCA ranks required to preserve variance over feature channels. Results are computed on 500 randomly sampled MEGABench examples.

Metric	Layer 0	Layer 10	Layer 20
Key rank 90%	1	15	38
Key rank 95%	1	37	57
Key rank 99%	3	89	97
Value rank 90%	102	99	100
Value rank 95%	114	113	113
Value rank 99%	125	125	125
Value rank 95% fraction	0.89	0.88	0.88
Value rank 99% fraction	0.98	0.98	0.98
Value top-10 ratio	0.28	0.32	0.26

To examine whether visual KV states can be compressed along the hidden dimension, we compute the cumulative PCA energy of keys and values over feature channels using 500 randomly sampled MEGABench examples. As shown in Figure 6 and Table 5, visual values are dense along the feature dimension: preserving 95% variance requires retaining 113-114 out of 128 dimensions across layers, and preserving 99% variance requires 125 dimensions. This makes hidden-dimension compression inefficient for visual values.

This analysis also distinguishes KVCapsule from MLA-style latent KV compression, which reduces KV memory by projecting keys and values into a compact latent feature space. In contrast, our results show that visual values are poorly compressible along the hidden dimension, while their sequence-level structure can still be exploited by PCA across vision tokens. Therefore, KVCapsule preserves the hidden dimension and compresses values along the vision token sequence, reducing memory without aggressively discarding feature-channel information.

C Calibration Procedure for Layer-wise KVCapsule Compression

KVCapsule uses one-time offline calibration of the visual-key retention masks, factorized key reconstructors, and value PCA bases while keeping the pretrained VLM frozen. For the reported setting, we collect uncompressed visual KV activations from 500 single-image calibration examples and calibrate the operators independently for each Transformer layer and KV head. The same calibrated operators are reused across all image and video datasets. The layer-wise pyramid schedule determines the retained sequence length r_ℓ, with the first $1/8$ of layers left uncompressed and the masking ratio increasing with depth to at most 0.85. For keys, the per-head masking logits and factorized reconstructor are optimized for 36 epochs using Adam with learning rate 10^{-5} and batch size 2. During the first 16 epochs, mask learning uses a differentiable rank-thresholded relaxation. Specifically, letting $p_{h,i} = \sigma(a_{h,i}/\tau)$ and q_h denote the r_ℓ-th largest entry of p_h, we use

$$\widetilde{m}_{h,i} = \sigma(20(p_{h,i} - q_h)) .$$

The first 6 epochs use Gaussian perturbation for exploration, followed by 10 focus epochs with a reduced temperature. The final 20 epochs use exact per-head top-r_ℓ selection. The calibration objective combines key-reconstruction MSE with regularization toward the target retention ratio and near-binary masks. For values, PCA is fitted independently for each layer and KV head along the visual sequence dimension using rank r_ℓ. After calibration, all masks, reconstruction factors, PCA bases, and centering vectors are fixed during inference. Algorithm 2 summarizes the procedure.

For completeness, FitValuePCA first computes the fixed calibration-set mean $\mu_{\ell,h}$ for each KV head, centers all calibration value activations using this mean, forms the sequence-position covariance $C_{\ell,h} \in \mathbb{R}^{n\times n}$, and retains the r_ℓ leading eigenvectors as $U_{\ell,h}$. Thus, PCA aggregates information across all calibration examples while preserving the visual position axis.

Algorithm 2 Offline Calibration of KVCapsule

Require: Layer-wise uncompressed visual KV activations $\{\mathcal{D}_\ell\}_{\ell=1}^{L}$, target retained lengths $\{r_\ell\}_{\ell=1}^{L}$
Ensure: Key masks $\{M_\ell\}$, key reconstruction factors $\{\theta_\ell = (W_{1,\ell}, W_{2,\ell})\}$, and value PCA bases $\{(U_\ell, \mu_\ell)\}$

for each layer ℓ **do**
  Initialize per-head masking logits α_ℓ
  Initialize factorized key reconstructor $\theta_\ell = (W_{1,\ell}, W_{2,\ell})$
  $(U_\ell, \mu_\ell) \leftarrow \textsc{FitValuePCA}(\mathcal{D}_\ell, r_\ell)$
  for $e = 0, \ldots, 35$ **do**
    for $K_\ell \sim \mathcal{D}_\ell$ **do**
      $M_\ell \leftarrow \textsc{CalibrateMask}(\alpha_\ell, r_\ell, e)$
      // CalibrateMask uses the differentiable rank-thresholded relaxation during the first 16 epochs and hard top-r_ℓ selection during the final 20 epochs.
      $\widetilde{K}_\ell \leftarrow \textsc{CompressKeys}(K_\ell, M_\ell)$
      $\widehat{K}_\ell \leftarrow W_{2,\ell} W_{1,\ell} \widetilde{K}_\ell$
      $\mathcal{L} \leftarrow \mathrm{MSE}(\widehat{K}_\ell, K_\ell) + 0.1\,\mathcal{L}_{\mathrm{mask}}$
      Update α_ℓ, $W_{1,\ell}$, and $W_{2,\ell}$ using Adam
    end for
  end for
  Fix M_ℓ using hard top-r_ℓ selection
end for

D Theoretical Memory Footprint of KVCapsule

We analyze the persistent memory footprint $\mathcal{S}$ of a single Transformer layer ℓ. For a standard Grouped-Query Attention (GQA) layer, the full-KV baseline stores keys and values for the text prompt, visual tokens, and generated tokens:

$$\mathcal{S}_{\mathrm{base}} = 2BH_{\mathrm{kv}}(c + n + t)d_{\mathrm{head}}b_{\mathrm{kv}}, \tag{5}$$

where B is the batch size, H_{kv} is the number of KV heads, c is the text-prompt length, n is the number of visual tokens, t is the generation step, d_{head} is the head dimension, and b_{kv} is the number of bytes per KV element.

KVCapsule reduces the stored visual sequence from n to $r_\ell = \lceil \ell_r n \rceil$. Its persistent footprint is

$$\mathcal{S}_{\mathrm{ours}} = \underbrace{2BH_{\mathrm{kv}}(c + r_\ell + t)d_{\mathrm{head}}b_{\mathrm{kv}}}_{\mathcal{S}_{\mathrm{request}}} + \underbrace{\mathcal{S}_{\mathrm{shared}}}_{\text{fixed operator state}}. \tag{6}$$

Here, $\mathcal{S}_{\mathrm{request}}$ is the request-specific KV cache, whereas $\mathcal{S}_{\mathrm{shared}}$ contains the calibrated operators shared across requests using the same visual-tokenization configuration.

For each layer, the factorized key reconstructor uses $\mathbf{W}_{1,\ell} \in \mathbb{R}^{H_{\mathrm{kv}} \times d_b \times r_\ell}$ and $\mathbf{W}_{2,\ell} \in \mathbb{R}^{H_{\mathrm{kv}} \times n \times d_b}$, while value compression uses $\mathbf{U}_\ell \in \mathbb{R}^{H_{\mathrm{kv}} \times r_\ell \times n}$ and $\boldsymbol{\mu}_\ell \in \mathbb{R}^{H_{\mathrm{kv}} \times d_{\mathrm{head}}}$. Thus,

$$\mathcal{S}_{\mathrm{shared}} = H_{\mathrm{kv}} d_b (n + r_\ell) b_W + H_{\mathrm{kv}} r_\ell n b_U + H_{\mathrm{kv}} d_{\mathrm{head}} b_\mu + \mathcal{S}_{\mathrm{sel}}, \tag{7}$$

where b_W, b_U, and b_μ denote the storage precision of the reconstruction factors, PCA basis, and centering vector, respectively, and $\mathcal{S}_{\mathrm{sel}}$ denotes the fixed selection metadata.

The key reconstruction factors therefore contain $H_{\mathrm{kv}} d_b (n + r_\ell)$ coefficients, compared with $H_{\mathrm{kv}} n r_\ell$ for a dense reconstruction map. The PCA basis contains $H_{\mathrm{kv}} r_\ell n$ coefficients. Unlike the request-specific KV cache, these operators are stored once per supported visual-tokenization configuration and do not scale with batch size or generated sequence length. Consequently, increasing context length or concurrent batch size amortizes the fixed operator state over larger KV-cache savings. Figure 5b shows the resulting memory-growth and break-even behavior for the evaluated configuration.

Measured memory breakdown. We additionally measure the persistent-memory components in the 10K system setting. Table 6 separates request-specific cache state from the calibrated operators shared across inputs with the same visual-tokenization configuration.

Table 6: Component-level persistent-memory accounting from the dedicated 10K memory-ledger run on one AMD MI250X.

Component	Memory (MB)	Scope
Full KV	1409.34	Per request
KVCapsule request state	630.58	Per request
KVCapsule operators	4.21	Shared
KVCapsule total	**634.79**	–
Reduction	**54.96%**	–

The shared operators occupy 4.21 MB and are reused across requests with the same visual-tokenization configuration. The component-level accounting yields a 54.96% reduction, consistent with the 55.0% reduction measured in the separate 10K efficiency sweep.

E Ablation on Key and Value Reconstruction Designs

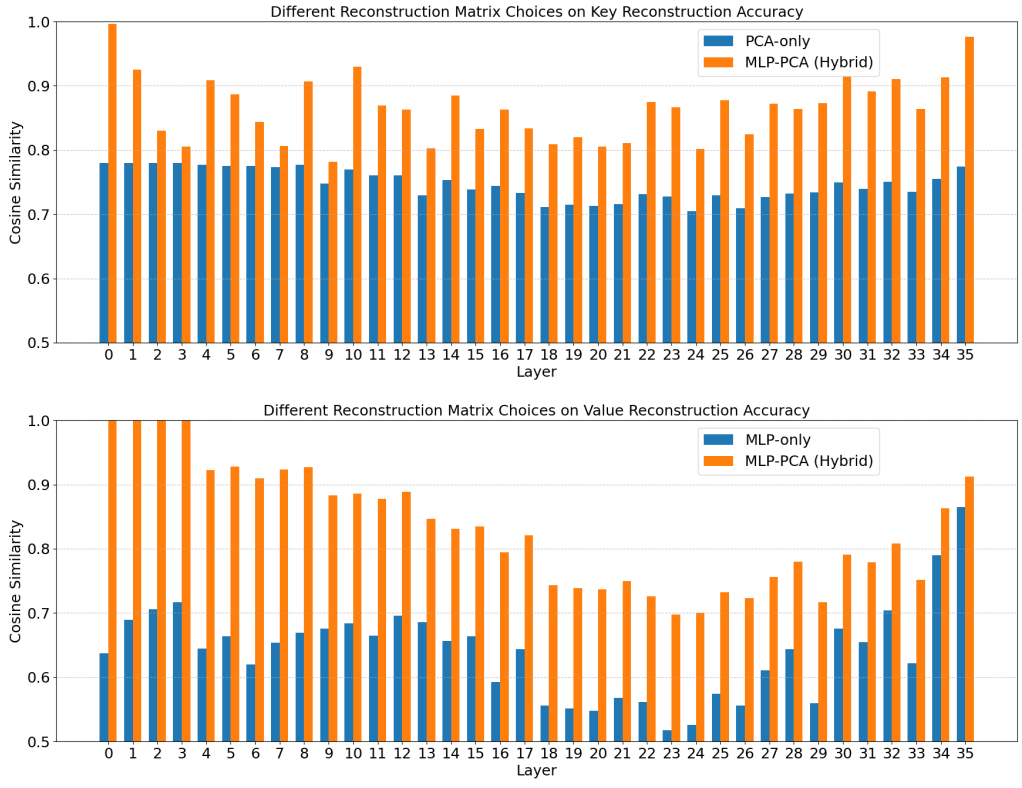


Figure 7: Key and value reconstruction designs. Cosine similarity between the original and reconstructed representations across all 36 layers (higher is better).

To validate the reconstruction design in KVCapsule, we conduct an ablation study using the `Qwen3-VL-8B-Instruct` model on 500 randomly selected samples from MEGABench Chen et al. (2025). The results compare a two-layer factorized linear reconstructor for keys with sequence-level PCA for values across all 36 layers. The factorized key reconstructor learns a cross-position mapping from the retained keys to the original visual sequence through an intermediate bottleneck of dimension $d_b = n/4$, while the PCA captures dominant sequence-level structure in the values.

The two branches therefore exploit different sequence structures while remaining fully linear. For keys, the factorized linear reconstruction introduces a structured intermediate representation and enables reconstruction to be algebraically fused with query-key computation. For values, PCA provides a compact sequence-level basis that preserves contributions from all visual positions. Their combination yields the asymmetric reconstruction design used by KVCapsule.

F Layer-wise Compression Ratio Sensitivity

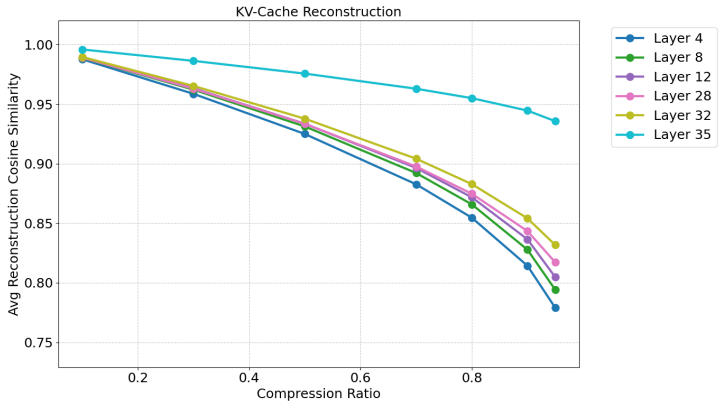


Figure 8: **KV Cache Reconstruction Accuracy vs. Compression Ratio.** The plot illustrates the relationship between compression ratio (mask ratio) and reconstruction cosine similarity across various model layers.

To determine how compression should be distributed across layers, we conduct a layer-wise sensitivity analysis on Qwen3-VL-8B-Instruct using 500 randomly selected MEGABench samples. We sweep representative layers over compression ratios from 0.1 to 0.95 and measure reconstruction fidelity using the mean cosine similarity between the original and reconstructed KV tensors.

Figure 8 shows clear layer-dependent differences in reconstruction sensitivity. Earlier and middle layers require higher retention to maintain reconstruction fidelity, while later layers tolerate more aggressive compression. For example, Layer 35 maintains reconstruction cosine similarity above 0.93 even at a 0.95 compression ratio.

Figure 4 further shows that visual KV states exhibit different inter-position similarity structures across model depth. Together, the two figures indicate that high inter-position similarity does not necessarily imply equal tolerance to compression across layers. We therefore use a pyramid compression schedule that assigns higher retention to more compression-sensitive layers and stronger compression to layers that maintain higher reconstruction fidelity.

G Ablation on K-side and V-side Reconstruction

The K-only / V-only ablation isolates how each component affects the final attention pattern. The results in Table 7 show that attention fidelity is more sensitive to K-side compression, since approximating keys directly changes the attention-score geometry and can shift where the model attends. In contrast, V-side compression appears less disruptive under attention-pattern metrics, but this does not mean values can be compressed aggressively: values carry the content aggregated by attention, so even small value perturbations can still degrade generation quality. This observation motivates KVCapsule's asymmetric design: we reconstruct keys to better preserve attention geometry, while applying a more conservative PCA-based compression to values instead of hard pruning. Overall, the ablation supports our claim that keys and values play distinct roles and should be compressed with different strategies rather than a single uniform compression rule.

Table 7: Ablation study on the individual effects of key and value reconstruction in KVCapsule.

Method	Cosine Similarity	MSE
KVCapsule (K only)	0.58	1.07×10^{-6}
KVCapsule (V only)	0.76	1.05×10^{-7}
KVCapsule	0.60	7.74×10^{-7}

H Ablation on Static Compression vs. Full-Position Reconstruction

We compare KVCapsule's full-position reconstruction with a variant that uses only the compressed positions directly in attention under the same persistent-cache setting. The reconstruction operators themselves are fixed after calibration; what changes across decoding steps is the query and therefore the attention assigned to the reconstructed visual positions. As shown in Table 8, restoring an approximate full visual-position representation substantially improves agreement with Full KV relative to attending only over the retained compressed positions. This result supports the benefit of recoverable position-space compression rather than irreversible position removal.

Table 8: Ablation of static compression versus full-position reconstruction. We compare the final attention weights against the full-cache baseline.

Comparison	Cosine Similarity	MSE
All KV vs. KVCapsule w/ reconstruction	0.60	7.74×10^{-7}
All KV vs. static KVCapsule w/o reconstruction	0.24	1.07×10^{-6}

I Image Resolution Sensitivity

Table 9 evaluates KVCapsule on Qwen3-VL under two image tokenization scales. In addition to the $1\times$ setting used in the main experiments, we evaluate the same images with approximately $4\times$ as many input pixels. The effect is task dependent: performance improves on several reasoning and hallucination benchmarks but decreases on COCO captioning. These results indicate that KVCapsule remains functional across changes in visual resolution, although the quality response is not uniform across tasks.

Table 9: Performance comparison of KVCapsule with different resolutions on VLMEvalKit.

	VLMs	MME	MMMU	COCO	MMBench	LLaVABench	HallusionBench
Qwen3	Baseline: Full KV	1735.60	0.52	15.70	0.80	74.80	53.90
	KVCapsule: 1x resolution	1708.49	0.48	15.80	0.78	73.80	53.70
	KVCapsule: $4\times$ pixel count	1740.78	0.54	13.12	0.84	76.80	63.40
	FastV	561.25	0.35	8.01	0.39	22.30	47.83

J Explicit and Fused KVCapsule

KVCapsule admits two equivalent inference paths. The explicit path reconstructs full-length visual K/V tensors before standard attention, while the fused path reassociates the same linear reconstruction operators to avoid materializing these tensors.

For keys, the factorized linear reconstructor is

$$\widehat{K} = W_2 W_1 \widetilde{K}, \tag{8}$$

where $W_1 \in \mathbb{R}^{d_b \times r}$ and $W_2 \in \mathbb{R}^{n \times d_b}$. We retain this factorized form rather than materializing a dense $n \times r$ reconstruction matrix. By associativity, the query–key product can be evaluated directly

Table 10: Explicit and fused KVCapsule. Stored visual KV reports the persistent visual cache tensors; both KVCapsule paths use the same compressed cache and shared reconstruction operators. Attention-time memory excludes shared backbone weights.

Path	Decode TPS	Stored visual KV cache (MB)	Attention-time memory (MB)
Full KV	16.6	1258	1613
Explicit KVCapsule	16.5	501	1613
Fused KVCapsule	**19.4**	501	**862**

Table 11: System efficiency at 10K context on one AMD MI250X. KV reduction is measured relative to Full KV.

Method	KV-related reduction	TPS
Full KV	0.0%	16.2
FastV	57.2%	22.0
PruMerge	46.1%	20.6
KVCapsule	**55.0%**	**20.1**
SparseVLMs	∼0%	16.1

in compressed score space as

$$\begin{aligned} Q\widehat{K}^\top &= Q\widetilde{K}^\top W_1^\top W_2^\top \\ &= \left((Q\widetilde{K}^\top)W_1^\top\right) W_2^\top. \end{aligned} \tag{9}$$

Similarly, for values reconstructed as

$$\widehat{V} = U^\top \widetilde{V} + \mathbf{1}_n \mu^\top, \tag{10}$$

the visual attention output can be evaluated as

$$P_{\text{img}}\widehat{V} = (P_{\text{img}}U^\top)\widetilde{V} + (P_{\text{img}}\mathbf{1}_n)\mu^\top. \tag{11}$$

Thus, fusion changes only the execution order and uses the same compressed cache and reconstruction operators as the explicit path.

As shown in Table 10, explicit reconstruction provides little runtime benefit because the full-length K/V tensors are materialized during attention. Explicit and fused KVCapsule use the same compressed visual KV cache, reducing stored visual KV by 60.2%. The fused path avoids materializing full-length K/V tensors during attention, reducing attention-time memory and improving decode throughput by 17.6% over explicit reconstruction.

K Measured Persistent-KV Efficiency

We compare realized KV-memory reduction and decoding throughput at 10K context on one AMD MI250X. Unlike visual-token reduction, the reported KV reduction measures persistent KV-related storage and includes uncompressed text KV, retained or compressed visual KV, indices, masks, rounding, and method-specific persistent states.

As shown in Table 11, KVCapsule reduces the persistent KV-related footprint by 55.0% and reaches 20.1 TPS. PruMerge achieves 46.1% reduction at 20.6 TPS, while FastV reaches 57.2% reduction and 22.0 TPS. FastV exhibits a larger quality drop on challenging video tasks; for example, its VCRBench score is 0.188 compared with 0.375 for KVCapsule. SparseVLMs sparsifies attention without physically compacting cached K/V tensors in our implementation and therefore provides little persistent-memory reduction.

Efficiency across compression levels. We further evaluate KVCapsule across a range of compression settings using the same 10K workload. Table 12 distinguishes the realized visual-KV reduction from the reduction in the complete persistent KV-related footprint.

Table 12: Memory and decoding efficiency across KVCapsule operating points at 10,014 input tokens (9,568 visual tokens) on one AMD MI250X. "Visual-KV red." denotes the realized visual-KV compression, while "Total red." measures the complete persistent KV-related footprint.

Setting	Visual-KV red. (%)	Persistent mem. (MB)	Total red. (%)	TPS
Full KV	–	1408.22	–	16.16
$c30$	29.7	1119.66	20.49	19.77
$c40$	39.6	970.85	31.06	19.86
$c50$	49.5	816.96	41.99	19.27
$c55$	55.1	734.20	47.86	19.55
$c60$	61.8	633.66	55.00	20.05
$c65$	66.7	560.71	60.18	19.24
$c70$	70.6	500.10	64.49	19.32

Table 13: Preservation of late-emerging visual information under matched retention. Higher is better.

Method	Late-important mass	Output cosine
Retention-only	0.32	0.36–0.51
KVCapsule	**0.63**	**0.80–0.82**

Increasing compression monotonically reduces the persistent-memory footprint. Decoding throughput remains above Full KV throughout the evaluated range and peaks near the $c60$ setting; more aggressive settings provide additional memory savings without further throughput improvement.

L Recoverability under Dynamic Visual Importance

To isolate the effect of full-position reconstruction, we compare KVCapsule with a retention-only variant using the same retained positions and the same layer-wise retention schedule. The retention-only variant performs attention directly over the retained positions without reconstructing the omitted visual positions. For each sample, layer ℓ, and decoding step s, we average Full-KV visual attention over query heads to obtain $a_{\ell,s,:}$. We define

$$\mathcal{I}^{\text{late}}_{\ell,s} = \text{TopK}_k(a_{\ell,s,:}) \setminus \text{TopK}_k(a_{\ell,0,:}), \qquad k = \max(1, \lceil 0.1 N_v \rceil),$$

where N_v is the number of visual positions. We measure the attention mass assigned to these late-important positions and the cosine similarity of the resulting attention outputs to Full KV.

As shown in Table 13, KVCapsule preserves substantially more information associated with late-important visual positions, increasing recovered attention mass from 0.32 to 0.63 and attention-output cosine from 0.36-0.51 to 0.80-0.82. These results confirm the benefit of reconstructing the full visual-position space when token importance shifts during decoding.

M Matched-Memory Ablation of Asymmetric K/V Compression

To isolate the effect of asymmetric K/V compression, we compare symmetric and asymmetric designs on Qwen3-VL-8B under the same 60.2% visual-KV reduction in Table 14.

At matched memory, the asymmetric design achieves the best compressed result on five of six metrics; symmetric PCA is slightly higher only on MMMU (0.486 vs. 0.480). Together with the measured K/V redundancy difference, this supports assigning learned reconstruction to keys and sequence-level PCA to values rather than applying either operator symmetrically.

N Compression-Rate Sensitivity

KVCapsule supports configurable compression through its layer-wise retention schedule. We evaluate a fixed family of calibrated configurations, denoted $c30$–$c70$, to characterize how system efficiency

Table 14: Matched-memory ablation of K/V compression on Qwen3-VL-8B. All compressed variants use 60.2% visual-KV reduction. Bold denotes the best compressed result for each metric.

Key method	Value method	MME	HallusionBench	MMBench	LLaVABench	MMMU	COCO	KV reduction
Full	Full	1735.7	53.90	0.800	74.8	0.520	15.70	0%
PCA	PCA	1479.2	48.32	0.746	73.0	**0.486**	12.11	60.2%
Learned recon.	Learned recon.	1672.0	50.07	0.764	68.8	0.457	14.79	60.2%
Learned recon.	**PCA**	**1708.5**	**53.70**	**0.780**	**73.8**	0.480	**15.80**	60.2%

and downstream quality change as compression increases. The complete memory-throughput sweep is reported in Table 12; here we report downstream quality at representative moderate, default, and aggressive settings.

Table 15: Quality-memory trade-off for representative configurations from the fixed KVCapsule compression sweep. "Visual-KV red." denotes realized visual-KV compression, while "Total red." denotes the complete persistent KV-related reduction measured in the 10K system workload. Quality results are evaluated on 500 randomly sampled examples from each benchmark, using the same samples across methods within each comparison.

Setting	Visual-KV red. (%)	Total red. (%)	MME	HBench	Video-MME
Full KV	–	–	1705.5	66.05	0.608
$c40$	39.6	31.06	1705.5	65.67	0.608
$c60$	61.8	55.00	1692.0	59.19	0.583
$c70$	70.6	64.49	1680.5	58.92	0.574

As shown in Table 15, the fixed sweep exhibits the expected quality–memory trade-off. At $c40$, KVCapsule reduces the complete persistent footprint by 31.1% while maintaining MME close to Full KV. The default $c60$ configuration increases the reduction to 55.0% with an MME score of 1692.0, while $c70$ provides further memory savings with a larger quality trade-off.

O Offline Calibration Overhead

KVCapsule does not fine-tune the pretrained VLM backbone. Instead, it performs one-time offline calibration of the auxiliary key reconstructor and value PCA basis from cached activations; all calibrated components are fixed during inference. Table 16 summarizes the calibration cost on one AMD MI250X.

Table 16: Offline calibration cost on one AMD MI250X.

Component	Samples	Calibration time
Factorized linear key reconstructor	500	$\sim$39 min
Value PCA fitting	500	$\sim$2 s
Total	**500**	**$\sim$39 min**

The calibration cost is dominated by key reconstruction, while PCA fitting is negligible. No backbone fine-tuning or per-request optimization is required.

We further evaluate calibration-sample efficiency on held-out examples. Table 17 reports reconstruction fidelity and output-level agreement as the calibration-set size varies.

50 calibration samples already provide high reconstruction fidelity, while increasing the calibration set to 400 samples further improves value reconstruction and output-level agreement. These results indicate that KVCapsule requires only modest one-time calibration rather than backbone fine-tuning.

Table 17: Effect of calibration-set size on held-out reconstruction and output fidelity.

Samples	Key cosine ↑	Value cosine ↑	Step-20 logit KL ↓	Top-1 agreement ↑
50	0.9266	0.767	0.95	0.82
400	0.9300	0.780	0.83	0.85

P Cross-Method Quality-Memory Trade-off

For the cross-method comparison, we group configurations by their realized persistent-memory footprint under the cross-method 10K system configuration. The memory-regime labels therefore denote measured persistent-KV reduction, rather than the visual-KV compression rate used to parameterize KVCapsule.

Table 18: Cross-method quality-efficiency trade-off at selected persistent-memory operating points under the cross-method 10K system configuration. Memory and reduction report the realized footprint of each evaluated configuration. Quality results are evaluated on 500 randomly sampled examples from each benchmark, using the same samples across methods within each comparison.

Persistent-memory regime	Method	KV Mem.	Persistent-KV red.	TPS	MME	HBench	V-MME	V-MMLU Cap.	V-MMLU QA
Ref.	Full KV	1414.6	0.0%	16.20	1713.5	70.05	0.658	0.196	0.334
∼40%	KVCapsule	824.7	41.7%	**19.70**	**1708.4**	**65.67**	**0.648**	**0.162**	**0.312**
	PruMerge	883.1	37.6%	19.57	1676.1	61.30	0.646	0.146	0.106
∼60%	KVCapsule	558.8	60.5%	**20.83**	**1672.0**	**59.19**	0.623	**0.095**	**0.300**
	PruMerge	609.4	56.9%	20.52	1671.9	48.16	**0.662**	0.092	0.064
∼70%	KVCapsule	424.4	70.0%	**21.80**	**1640.5**	**56.92**	**0.654**	**0.108**	**0.288**
	PruMerge	479.0	66.1%	21.77	1639.1	48.16	0.639	0.101	0.069

As shown in Table 18, across the selected memory regimes, KVCapsule and PruMerge exhibit complementary quality-efficiency trade-offs. Around 40% reduction, KVCapsule achieves higher scores across all reported quality metrics while providing comparable throughput. Around 60%, KVCapsule retains higher HallusionBench and Video-MMLU performance, while PruMerge achieves higher MME and Video-MME. At the most aggressive operating point, KVCapsule reaches 70.0% persistent-memory reduction and 21.80 TPS, while outperforming PruMerge on HallusionBench, Video-MME, and both Video-MMLU tasks; PruMerge retains higher MME.

The multi-rate sweep focuses on KVCapsule and PruMerge, for which we evaluate corresponding operating points across all three memory budgets. FastV is reported separately at its evaluated operating point in Table 11. SparseVLMs is not included because its evaluated implementation sparsifies attention without physically compacting cached K/V tensors and therefore does not expose comparable persistent-memory reduction operating points.

Q Support for Variable Visual Contexts

KVCapsule uses a reusable bank of offline-calibrated operators for the visual-token lengths produced by the vision encoder. For an image or sampled video frame represented by n visual tokens, with compressed length r and key-reconstruction bottleneck dimension d_b, the key reconstruction factors and value projection operator have dimensions

$$W_1 \in \mathbb{R}^{d_b \times r}, \qquad W_2 \in \mathbb{R}^{n \times d_b}, \qquad U \in \mathbb{R}^{r \times n}. \tag{12}$$

Thus, the key reconstruction mapping is represented using $d_b(n+r)$ coefficients rather than explicitly storing a dense $n \times r$ operator. Inputs sharing the same visual-tokenization configuration reuse the same masks, reconstruction factors, and PCA bases. Additional images or video frames therefore reuse the corresponding fixed operators rather than increasing their dimensions. No operator fitting, interpolation, or resizing is performed during inference; an unseen configuration requires one-time offline calibration.

Table 19 evaluates KVCapsule across visual contexts ranging from 8 to 64 video frames. KVCapsule remains close to Full KV on LongVideoBench and matches the reported Full-KV scores on MLVU and Video-MMLU QA. Appendix I further evaluates robustness across image resolutions.

Table 19: Performance across different visual-context lengths.

Dataset	Frames	Full KV	KVCapsule
LongVideoBench	8	0.598	0.594
LongVideoBench	64	0.658	0.646
MLVU	8	0.576	0.576
Video-MMLU QA	16	0.390	0.390

Memory scaling with video length. We evaluate persistent memory while increasing the number of video frames under a fixed per-frame tokenization configuration. The same calibrated operators are reused across frames, while request-specific visual KV grows with the visual context.

Table 20: Persistent-memory scaling with video length.

Frames	Full KV (MB)	KVCapsule (MB)	Reduction
1	68.63	64.37	6.2%
8	172.97	109.46	36.7%
16	292.22	160.99	44.9%
32	530.72	264.06	50.2%
64	1007.72	470.18	53.3%

The memory reduction increases with visual-context length, from 6.2% for one frame to 53.3% for 64 frames, as the fixed shared state is amortized over a larger visual KV cache.

R Output-Level Fidelity of K/V Compression

The K-only/V-only ablation in Table 7 measures downstream attention-pattern fidelity. Keys directly affect the current-layer attention weights through $\mathrm{softmax}(QK^\top)$. Values do not alter the same-layer softmax directly, but value perturbations change the resulting hidden states and can therefore modify queries and attention patterns in subsequent layers and decoding steps. Consequently, attention-weight fidelity is expected to be more directly sensitive to key compression, while value compression should additionally be assessed through attention-output and generation-level metrics.

To complement the attention-weight analysis, we evaluate output-level fidelity on a fixed deterministic 60-example LLaVABench subset at the paper's compression operating point. Table 21 reports attention-output SNR, logit KL divergence, top-1 prediction agreement, and perplexity relative to Full KV.

Table 21: Output-level fidelity of K/V compression. Logit KL and top-1 agreement are reported from decoding steps 1 to 20. Full KV serves as the reference, so attention-output SNR is not applicable to that row.

Method	Attn. output SNR	Logit KL	Top-1 agree.	PPL / Full KV	KV reduction
Full KV	–	–	1.00	1.00×	0%
V-only, PCA value	22–23 dB	1.83→0.77	0.55→0.82	2.79×	30.1%
K-only, reconstructed key	19–20 dB	1.92→0.94	0.52→0.87	2.23×	30.1%
KVCapsule	18–20 dB	2.06→0.80	0.45→0.85	2.23×	60.2%

As shown in Table 21, K-only compression and KVCapsule obtain the same sequence-level perplexity (2.23×), despite KVCapsule compressing both K and V and therefore achieving substantially greater KV reduction. KVCapsule also reduces the step-20 logit KL from 0.94 to 0.80 relative to K-only, while maintaining comparable top-1 agreement. Thus, the value branch provides additional memory reduction without further degrading sequence-level perplexity, supporting asymmetric compression of both cache components rather than key-only compression.

Together with the matched-memory downstream ablation in Table 14, these results support the distinct roles of the two branches: key reconstruction primarily preserves query–key geometry, whereas value

compression should be assessed after attention aggregation rather than through attention-weight similarity alone.