REVISKV: CONVERTING DYNAMIC VISUAL EXECUTION SPARSITY INTO REVERSIBLE GPU KV RESIDENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models repeatedly access visual evidence during extended reasoning, making the visual key-value (KV) cache a persistent source of GPU memory pressure. Dynamic visual selection reduces attention computation, but retaining complete candidate KV on GPU leaves this storage cost unchanged. We introduce ReVisKV, which preserves complete visual KV in CPU memory for managed decoder layers and maintains only a small hot buffer on GPU, recalling missing entries on demand. Motivated by cross-layer drift in visual working sets, ReVisKV refreshes selections at multiple anchors, allowing offloading to begin early while preserving access to later-needed evidence. The hot buffer exploits short-term reuse across decoding steps to reduce transfers, and stage-shared cache decisions with asynchronous execution reduce management overhead. On Qwen3-VL-Thinking 4B, ReVisKV retains 99.5% of VisionPulse's average score across five benchmarks at a 10% selection upper bound, while using 62.5% less GPU visual-KV capacity. Compared with dense decoding, the optimized runtime delivers up to $1.37\times$ decode speedup and supports up to $2.5\times$ the batch size.

1 INTRODUCTION

Vision-language models combine high-resolution perception with extended reasoning to interpret documents, charts, and complex scenes (Bai et al., 2025; Wang et al., 2025). A detailed image can contribute thousands of visual tokens, whose key-value (KV) states remain available throughout generation. This persistent context lets the model revisit an image as its answer develops, but also occupies GPU memory at every decoder layer. Higher input resolution increases the cache held for each request, leaving less capacity for additional requests and longer outputs.

Visual token reduction exploits redundancy at different stages in inference. Encoder-side compression condenses visual features before language processing (Yang et al., 2025), while decoder-side pruning uses intermediate attention to reduce subsequent computation (Chen et al., 2024; Xing et al., 2025). Selection can account for visual and textual coverage (Dong et al., 2026) or preserve spatial structure during compression (Huang et al., 2026). These criteria improve the information retained by an early selection. However, extended reasoning changes which evidence is needed over time. VisionPulse (Xu et al., 2026) responds by selecting a new visual subset at each decoding step, allowing attention to revisit previously omitted regions while retaining the complete candidate KV on GPU.

Sparse visual attention therefore leaves a storage bottleneck. Each request still occupies GPU memory for visual KV that may be inactive for many steps, limiting the number of requests that fit concurrently. Permanent deletion releases this memory but prevents later reconsideration of discarded evidence. Host-backed retrieval separates accessibility from GPU residency, at the cost of retrieving missing KV (Lee et al., 2024; Sun et al., 2025). For visual reasoning, the challenge is to maintain a small GPU working set while preserving access to the complete visual context, refreshing selections across depth and recalling changes efficiently.

We examine how these working sets change across both decoder depth and generation time. The diagonal structure in Figure 1(a) indicates that neighboring layers select similar visual evidence,

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.

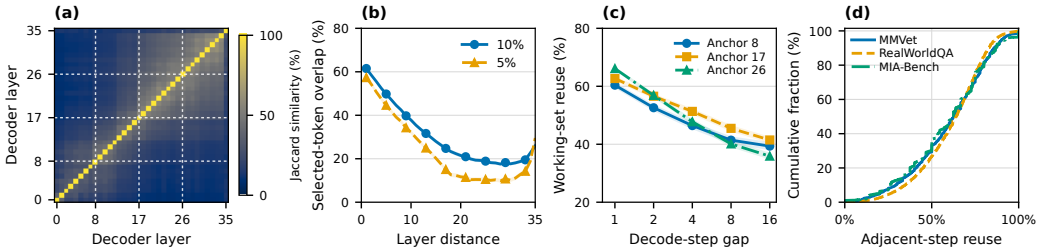

Figure 1: **Visual working sets drift across layers but persist across nearby decoding steps.** (a) Layer-pair Jaccard similarity; (b) token overlap versus layer distance; (c) working-set reuse versus decode-step gap; (d) cumulative distribution of adjacent-step reuse, pooled over anchor-step observations.

whereas distant layers often favor different regions. Figure 1(b) quantifies this drift. At 10% retention, overlap decreases from 61.5% between adjacent layers to 26.0% across the full decoder depth. A selection made near the beginning of the decoder therefore cannot represent all later layers equally well, motivating refresh at multiple depths.

Temporal overlap, however, offers an opportunity to make refresh inexpensive. Figure 1(c) shows that nearby decoding steps reuse more of the same working set than distant steps. In Figure 1(d), median adjacent-step reuse is approximately 63% to 64% on MMVet (Yu et al., 2024), RealWorldQA (xAI, 2024), and MIA-Bench (Qian et al., 2025). Thus, much of a new selection is already available from the preceding step, even though the selection changes over a longer sequence. Together, these observations motivate refreshing relevance across depth while caching reusable KV across time.

We introduce **ReVisKV** to convert dynamic visual sparsity into reversible GPU KV residency. Each managed layer keeps its complete visual KV in CPU memory and uses a bounded GPU hot buffer that fetches only missing entries. Multiple anchors refresh working sets across decoder depth, enabling offloading to begin early while later layers can revise their visual focus. Reusing resident KV across steps reduces transfers, and shared cache decisions within each stage limit management overhead. On Qwen3-VL-Thinking 4B, ReVisKV retains 99.5% of VisionPulse's average score across five benchmarks with a 10% selection upper bound and 62.5% less GPU visual-KV capacity. Compared with Dense, the optimized runtime delivers up to $1.37\times$ decode speedup and supports up to $2.5\times$ the maximum batch size. Our vLLM integration further demonstrates improved responsiveness under high request concurrency.

Our contributions are:

- **A characterization of visual working-set dynamics.** We examine relevance across decoder depth and decoding time, identifying cross-layer drift and short-term reuse as complementary design signals for visual-KV residency.
- **Multi-anchor reversible visual-KV offloading.** We combine complete CPU backing with bounded GPU hot buffers and stage-wise relevance refresh, reducing GPU storage while preserving access to inactive visual evidence.
- **Temporal reuse and stage-shared recall.** Cross-step hot-buffer reuse reduces KV transfers, while shared cache decisions, GPU kernels, and asynchronous submission reduce management and synchronization overhead. Quality and system experiments evaluate the resulting trade-offs across backbones, selectors, and workloads.

2 Related Work

Visual token compression and pruning. Visual compression reduces the representation entering the decoder, as in VisionZip (Yang et al., 2025), or the tokens processed by later layers, as in FastV (Chen et al., 2024) and PyramidDrop (Xing et al., 2025). Coverage-based selection (Dong et al., 2026) and spatially structured reduction (Huang et al., 2026) preserve informative content. However, fixing the available evidence before decoding prevents later steps from reconsidering discarded details. Aggressive compression can therefore conflict with changing evidence needs during extended reasoning (Xu et al., 2026).

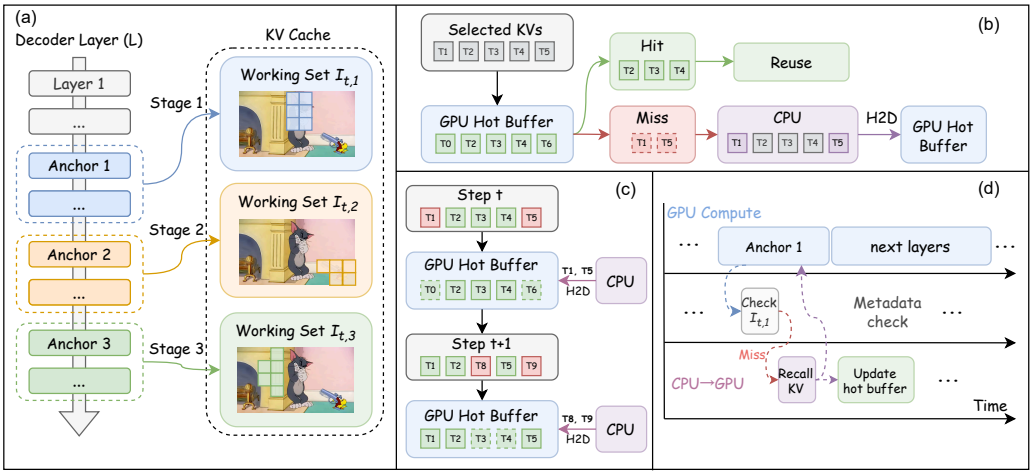

Figure 2: **ReVisKV overview.** (a) Anchors refresh working sets across decoder stages; (b) hot-buffer hits reuse GPU KV and misses trigger recall from exact CPU backing; (c) successive steps transfer only missing entries; (d) CPU recall preparation proceeds alongside queued GPU computation.

KV eviction and adaptive allocation. KV eviction uses attention sinks, recency, or attention statistics to decide which entries to retain (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024). Capacity can vary across layers or heads (Cai et al., 2024; Qin et al., 2025; Feng et al., 2025), while reconstruction and spatiotemporal smoothing refine retention scores (An et al., 2026). VisCache (Wang et al., 2026) combines keyframe selection with KV pruning and value fusion. These eviction policies improve the choice of retained information, but entries removed without a recoverable copy are permanently unavailable. A low score at one step can therefore prevent access when the same evidence becomes relevant later.

Dynamic selection and host-backed retrieval. Query-dependent access underlies page selection in Quest (Tang et al., 2024) and step-wise visual selection in VisionPulse (Xu et al., 2026). Host-backed methods schedule computation and storage across memory tiers (Sheng et al., 2023), predict which KV entries will be useful (Lee et al., 2024), or combine low-rank keys with offloaded values and temporal reuse (Sun et al., 2025). HybridKV (Zeng et al., 2026) uses text-centric attention to classify heads and allocates hierarchical budgets, pruning static heads and offloading dynamic-head KV to CPU for chunk-wise retrieval. VisionPulse retains candidate visual KV on GPU for reselection, so reducing its active subset does not release the underlying storage. ReVisKV couples full-context stage refresh with reusable GPU hot buffers, jointly managing which visual evidence is selected and which KV remains resident or is recalled from CPU.

3 REVISKV

ReVisKV combines complete CPU backing with bounded GPU hot buffers to reduce managed-layer visual-KV residency. Figure 2(a) shows multi-anchor selection across depth. The GPU hot buffer in Figure 2(b) serves selected KV from resident slots or CPU memory. Figure 2(c) illustrates reuse across steps, and Figure 2(d) shows the execution schedule.

3.1 MULTI-ANCHOR VISUAL WORKING-SET REFRESH

Let $\mathcal{V}$ be the set of input visual-token IDs, with $N_v = |\mathcal{V}|$, and let t index decoding steps. Prefill constructs full visual KV at every layer. During decoding, anchors $a_1 < \cdots < a_M$ partition the decoder into managed stages. Stage j contains the layers strictly after a_j and before the next anchor, or through the final layer for the last stage. The prefix through a_1 and all later anchors retain full visual KV on GPU, so each anchor can assess evidence omitted by earlier stages.

As illustrated in Figure 2(a), anchor a_j selects a working set $I_{t,j} \subseteq \mathcal{V}$ using the current decoding query. Its managed layers attend to their own KV for these IDs. The next anchor reassesses the full candidate set using its updated hidden state, allowing a later stage to focus on different

image regions. This limits the depth over which one selection is reused and addresses the drift in Figure 1(a) and (b). An early anchor enables offloading in more layers, while later anchors relieve it of having to anticipate every subsequent use of the image. Text and generated-token KV remain under the backbone's standard cache policy.

We rank visual IDs using the attention-based estimator of FastV (Chen et al., 2024) and VisionPulse (Xu et al., 2026). Let $A_{t,h}^{(a_j)}(i)$ denote the attention probability assigned to visual token i by query head h, with H query heads. The head-averaged score determines the selected IDs,

$$s_{t,j}(i) = \frac{1}{H}\sum_{h=1}^{H} A_{t,h}^{(a_j)}(i), \qquad I_{t,j} = \mathrm{TopK}_{k_{t,j}}\big(\{s_{t,j}(i) : i \in \mathcal{V}\}\big), \tag{1}$$

where TopK returns token IDs and $k_{t,j}$ is the selected-token count. Attention probabilities are normalized over the full causal context, including visual and text tokens.

To adapt the count to the current need for visual evidence, we use VisionPulse's visual-mass budget (Xu et al., 2026). Let $A_{t,h}^{(a_j,\tau)}$ be the attention distribution obtained by applying softmax to the anchor's logits divided by temperature $\tau > 0$. Its largest visual mass across heads determines the selected fraction within bounds $0 < r_{\min} \le r_{\max} \le 1$,

$$m_{t,j} = \max_{1\le h\le H} \sum_{i\in\mathcal{V}} A_{t,h}^{(a_j,\tau)}(i), \qquad k_{t,j} = \lceil N_v \ \mathrm{clip}(m_{t,j}, r_{\min}, r_{\max}) \rceil . \tag{2}$$

Here clip bounds its first argument to $[r_{\min}, r_{\max}]$, with both fractions measured against N_v. Temperature scaling controls the count, while Equation 1 ranks IDs using the original probabilities. ReVisKV applies these existing selection rules at multiple anchors and connects their decisions to reversible storage, as described next.

3.2 Reversible offloading and the GPU hot buffer

Figure 2(b) separates the active working set from the KV physically resident on GPU. After prefill, each managed layer preserves its full visual KV in pinned CPU memory at the original precision and releases the full GPU copy. A bounded GPU hot buffer then holds the selected KV together with recently used entries. Because visual KV is immutable during decoding, replacing a GPU entry requires no write-back, and its CPU copy remains available for later recall.

Selected entries already in the hot buffer are hits and are reused in place. Misses trigger host-to-device (H2D) recall from CPU memory into available GPU slots before attention executes. In the example in Figure 2(b), T_2, T_3, and T_4 are hits, so only T_1 and T_5 cross the CPU–GPU boundary. Attention uses the selected visual entries together with causal text KV. Unselected entries may remain cached for reuse but do not participate in attention.

A later anchor or decoding step can select any original visual ID, including one whose managed-layer GPU copies have been replaced. Full-context anchors preserve this ability to reconsider the image, and exact CPU backing supplies the corresponding layer-specific KV. Reversibility therefore preserves access to evidence while attention itself remains sparse.

The default 36-layer layout uses zero-based anchors $[8, 17, 26]$ and a hot buffer accommodating 10% of input visual tokens at each managed layer. Including the full-resident prefix and anchors, it allocates 37.5% of dense visual-KV capacity on GPU. This storage fraction differs from the active-token fraction and total GPU memory.

3.3 Temporal reuse and incremental recall

The hot buffer reduces H2D traffic by retaining entries across changes in the working set. Let $C_{t-1,j}$ denote the resident visual IDs before stage j refreshes at step t. Only selected IDs absent from this set require recall,

$$\Delta I_{t,j} = I_{t,j} \setminus C_{t-1,j}. \tag{3}$$

The initially empty buffer is populated by the first selection. On subsequent steps, entries in $I_{t,j} \cap C_{t-1,j}$ remain in place, while misses occupy free slots or replace least-recently-used entries outside the current selection. Protecting all selected entries ensures that every managed layer can attend to

the complete working set. The updated resident set $C_{t,j}$ contains $I_{t,j}$ and any history that fits in the remaining capacity.

Figure 2(c) illustrates why recall is incremental. After T_1 and T_5 are fetched at step t, both remain available at step $t+1$, so only the newly requested T_8 and T_9 need H2D transfer. More generally, a token can leave the active set and return without transfer while its slot remains cached. If that slot has been replaced, the token can still return through CPU recall. The hot buffer exploits short-term reuse, while CPU backing preserves access across longer absences. Increasing buffer capacity retains more history without enlarging the active attention set.

3.4 Stage-shared recall execution

Stage-wise selection also reduces recall-management work. All layers in a stage receive the same selected IDs and, given identical cache initialization, capacity, and replacement rules, follow the same residency transitions. ReVisKV therefore computes hit, miss, and slot-assignment decisions once per request and stage, while keeping each layer's KV values in a separate preallocated hot buffer.

Figure 2(d) shows the execution dependencies. Once an anchor produces $I_{t,j}$, the CPU checks the shared resident-ID map and prepares the miss list and slot assignments. These metadata are uploaded asynchronously. For each managed layer, a Triton recall kernel gathers only missing KV from mapped pinned CPU memory into the assigned GPU slots. A second Triton kernel translates the shared slot assignments into layer-specific addresses for sparse attention. This makes one stage-level cache decision usable by every managed layer without reconstructing its address table on CPU.

Recall, address construction, and attention execute in GPU stream order, so attention reads each slot only after its KV is ready. Meanwhile, asynchronous submission lets the CPU prepare and enqueue subsequent layer work while the GPU executes earlier operations. This removes a CPU wait at each managed layer without relaxing the dependency on the anchor's current selection. Algorithm 1 in Appendix A gives the corresponding cache updates.

4 Experiments

4.1 Experimental setup

We evaluate Qwen3-VL-Thinking 4B on MMVet (Yu et al., 2024), MIA-Bench (Qian et al., 2025), RealWorldQA (xAI, 2024), InfoVQA (Mathew et al., 2022), and ChartQA (Masry et al., 2022), covering integrated visual reasoning, instruction following, scene understanding, and document or chart interpretation. Baselines include dense decoding, VisionZip (Yang et al., 2025), VisionPulse (Xu et al., 2026), and a HybridKV reimplementation (Zeng et al., 2026). HybridKV retains its visual KV on GPU for this quality comparison. Methods share preprocessing, generation settings, and scoring rules within each backbone. We report task scores on a 0–100 scale, their unweighted mean, and mean generated length. Quality evaluation allows generation to end naturally, subject to an 8192-token cap.

The main comparisons use 10% and 5% visual-token budgets. For VisionPulse and ReVisKV, these are dynamic upper bounds, paired with lower bounds of 5% and 1%, respectively. ReVisKV uses anchors $[8, 17, 26]$ and a 10% hot buffer unless varied explicitly. System experiments use one NVIDIA A800-80GB with matched inputs and fixed output lengths to separate per-step efficiency from changes in generation length. We measure host-to-device (H2D) KV traffic, GPU memory, time per output token (TPOT), end-to-end latency, and aggregate throughput.

4.2 Quality under aggressive visual budgets

Table 1 shows that ReVisKV retains 99.0% and 97.7% of Dense's average score at selection upper bounds of 10% and 5%. At 10%, it preserves over 97% of the Dense score on every task, including over 99% on MIA-Bench, RealWorldQA, and InfoVQA. The quality benefit thus extends across instruction following, scene understanding, and document interpretation.

Table 1: **Quality and output length on Qwen3-VL-Thinking 4B.** L denotes mean generated tokens and S the task score. Absolute values appear above their relative changes from Dense. Avg is the unweighted mean across datasets.

Method	MMVet		MIA-Bench		RealWorldQA		InfoVQA		ChartQA		Avg	
	$L\downarrow$	$S\uparrow$	$L\downarrow$	$S\uparrow$	$L\downarrow$	$S\uparrow$	$L\downarrow$	$S\uparrow$	$L\downarrow$	$S\uparrow$	$L\downarrow$	$S\uparrow$
Full visual tokens (100%)												
Dense	1313.4	70.37	1846.9	91.21	606.6	72.16	480.9	83.11	397.6	78.52	929.1	79.07
10% visual-token budget												
VisionZip	1727.8	40.69	2005.6	87.95	541.8	56.99	1651.4	24.00	1007.8	21.76	1386.9	46.28
	+31.5%	*-42.2%*	*+8.6%*	*-3.6%*	*-10.7%*	*-21.0%*	*+243.4%*	*-71.1%*	*+153.5%*	*-72.3%*	*+49.3%*	*-41.5%*
HybridKV	1958.1	61.10	2133.2	88.16	533.0	71.76	826.1	77.79	1001.1	73.80	1290.3	74.52
	+49.1%	*-13.2%*	*+15.5%*	*-3.3%*	*-12.1%*	*-0.5%*	*+71.8%*	*-6.4%*	*+151.8%*	*-6.0%*	*+38.9%*	*-5.8%*
VisionPulse	1264.4	70.09	1796.9	91.14	518.2	72.42	510.2	82.54	441.5	77.48	906.2	78.73
	-3.7%	*-0.4%*	*-2.7%*	*-0.1%*	*-14.6%*	*+0.4%*	*+6.1%*	*-0.7%*	*+11.0%*	*-1.3%*	*-2.5%*	*-0.4%*
ReVisKV (Ours)	1219.8	68.44	1750.5	90.83	489.8	72.03	479.5	82.55	420.7	77.68	872.1	78.31
	-7.1%	*-2.7%*	*-5.2%*	*-0.4%*	*-19.3%*	*-0.2%*	*-0.3%*	*-0.7%*	*+5.8%*	*-1.1%*	*-6.1%*	*-1.0%*
5% visual-token budget												
VisionZip	1553.5	30.05	2057.2	83.81	522.6	49.15	978.2	22.57	704.4	15.48	1163.2	40.21
	+18.3%	*-57.3%*	*+11.4%*	*-8.1%*	*-13.8%*	*-31.9%*	*+103.4%*	*-72.8%*	*+77.2%*	*-80.3%*	*+25.2%*	*-49.1%*
HybridKV	1862.8	57.20	2191.9	88.13	542.2	72.94	1037.7	74.22	1353.1	69.84	1397.5	72.47
	+41.8%	*-18.7%*	*+18.7%*	*-3.4%*	*-10.6%*	*+1.1%*	*+115.8%*	*-10.7%*	*+240.3%*	*-11.1%*	*+50.4%*	*-8.4%*
VisionPulse	1285.8	67.29	1788.6	91.09	463.3	73.07	521.1	81.48	534.0	76.24	918.6	77.83
	-2.1%	*-4.4%*	*-3.2%*	*-0.1%*	*-23.6%*	*+1.3%*	*+8.4%*	*-2.0%*	*+34.3%*	*-2.9%*	*-1.1%*	*-1.6%*
ReVisKV (Ours)	1321.4	66.28	1835.5	90.91	495.3	71.50	535.7	81.10	565.0	76.40	950.6	77.24
	+0.6%	*-5.8%*	*-0.6%*	*-0.3%*	*-18.3%*	*-0.9%*	*+11.4%*	*-2.4%*	*+42.1%*	*-2.7%*	*+2.3%*	*-2.3%*

Table 2: **Quality across backbones.** S denotes the task score and Δ its relative change from Dense on the same backbone. Avg is the unweighted mean of the two task scores.

Backbone	Method	Budget	MMVet		RealWorldQA		Avg	
		(%)	$S\uparrow$	Δ	$S\uparrow$	Δ	$S\uparrow$	Δ
Qwen3-VL-Thinking 8B	**Dense**	100	66.88	–	73.20	–	70.04	–
	VisionPulse	10	64.13	*-4.1%*	71.24	*-2.7%*	67.69	*-3.4%*
	ReVisKV	10	65.55	*-2.0%*	70.59	*-3.6%*	68.07	*-2.8%*
	VisionPulse	5	64.68	*-3.3%*	72.94	*-0.4%*	68.81	*-1.8%*
	ReVisKV	5	63.81	*-4.6%*	71.11	*-2.9%*	67.46	*-3.7%*
InternVL-3.5-Thinking 4B	**Dense**	100	61.70	–	48.63	–	55.16	–
	VisionPulse	10	62.34	*+1.0%*	51.76	*+6.5%*	57.05	*+3.4%*
	ReVisKV	10	63.35	*+2.7%*	49.93	*+2.7%*	56.64	*+2.7%*
	VisionPulse	5	62.52	*+1.3%*	51.24	*+5.4%*	56.88	*+3.1%*
	ReVisKV	5	64.04	*+3.8%*	52.94	*+8.9%*	58.49	*+6.0%*

ReVisKV exceeds VisionZip on every task at both budgets, and HybridKV on all five tasks at 10% and four at 5%. VisionZip loses 71.1% to 80.3% of the Dense score on InfoVQA and ChartQA, whereas ReVisKV's losses remain below 3% on both tasks. HybridKV is less affected overall but still loses 18.7% on MMVet at 5% retention. These differences highlight the difficulty of preserving later-needed visual evidence under aggressive compression.

ReVisKV also retains 99.5% and 99.2% of VisionPulse's average score at the two upper bounds. VisionPulse keeps the complete candidate visual KV on GPU, requiring 2.67× the visual-KV capacity of ReVisKV's default layout. ReVisKV therefore preserves comparable quality with 62.5% less visual-KV allocation, freeing GPU capacity for additional requests.

ReVisKV's mean output is 6.1% shorter than Dense at the 10% upper bound and 2.3% longer at 5%. VisionZip instead lengthens outputs by 49.3% and 25.2% on average, with a 243.4% increase on InfoVQA at 10%. Inspection of its InfoVQA and ChartQA responses confirms cases of repeated words or phrases continuing to the generation limit without a correct answer. Thus, reduced attention computation can be offset by longer, unsuccessful generations when the retained evidence is insufficient.

4.3 Transfer across backbones and selectors

Table 2 shows that ReVisKV achieves 100.6% and 98.0% of VisionPulse's mean score on Qwen3-VL-Thinking 8B at the 10% and 5% upper bounds. Both additional backbones have 36 decoder layers, for which ReVisKV's default layout requires 62.5% less GPU visual-KV capacity. On InternVL-3.5-Thinking 4B, ReVisKV exceeds Dense's mean score by 2.7% and 6.0%. At 5%, it

Table 3: **Compatibility with alternative visual selectors.** Native prunes during prefill, whereas +ReVisKV adapts each selector to reversible decoding. S denotes the task score and Δ its relative change from Dense. Avg is the unweighted mean of the three task scores.

Selector	Variant	Budget (%)	MMVet $S\uparrow$	MMVet Δ	MIA-Bench $S\uparrow$	MIA-Bench Δ	RealWorldQA $S\uparrow$	RealWorldQA Δ	Avg $S\uparrow$	Avg Δ
Dense	–	100	70.37	–	91.21	–	72.16	–	77.91	–
FastV	Native	10	41.70	*-40.7%*	84.02	*-7.9%*	61.83	*-14.3%*	62.51	*-19.8%*
	+**ReVisKV**	10	68.44	*-2.7%*	90.83	*-0.4%*	72.03	*-0.2%*	77.10	*-1.0%*
	Native	5	36.42	*-48.2%*	83.21	*-8.8%*	61.96	*-14.1%*	60.53	*-22.3%*
	+**ReVisKV**	5	66.28	*-5.8%*	90.91	*-0.3%*	71.50	*-0.9%*	76.23	*-2.2%*
PDrop	Native	10	41.83	*-40.5%*	80.33	*-11.9%*	69.28	*-4.0%*	63.81	*-18.1%*
	+**ReVisKV**	10	67.98	*-3.4%*	90.91	*-0.3%*	71.11	*-1.4%*	76.67	*-1.6%*
	Native	5	40.09	*-43.0%*	82.10	*-10.0%*	64.97	*-10.0%*	62.38	*-19.9%*
	+**ReVisKV**	5	67.48	*-4.1%*	90.86	*-0.4%*	72.94	*+1.1%*	77.09	*-1.1%*

Table 4: **Effect of anchor number and placement.** Scores (S) use fixed 10% retention, while system costs use dynamic selection bounded between 5% and 10%. Δ is the relative score change from Dense, and Avg is the unweighted mean of the two task scores. Bold marks the best sparse scores.

Anchors	#	Residency (%) ↓	H2D (MB/token) ↓	MMVet $S\uparrow$	MMVet Δ	RealWorldQA $S\uparrow$	RealWorldQA Δ	Avg $S\uparrow$	Avg Δ
Dense	–	100.0	0.0	70.37	–	72.16	–	71.26	–
[8]	1	32.5	8.4	60.28	*-14.3%*	71.90	*-0.4%*	66.09	*-7.3%*
[17]	1	55.0	11.9	68.17	*-3.1%*	71.90	*-0.4%*	70.03	*-1.7%*
[8, 17]	2	35.0	15.2	66.51	*-5.5%*	71.50	*-0.9%*	69.01	*-3.2%*
[17, 26]	2	57.5	11.4	67.06	*-4.7%*	72.03	*-0.2%*	69.55	*-2.4%*
[**8**, **17**, **26**]	3	37.5	13.9	**68.58**	*-2.5%*	**73.07**	*+1.3%*	**70.82**	*-0.6%*

also outperforms VisionPulse on both tasks, improving the mean by 2.8%. Reversible working sets thus preserve quality across these model families.

Table 3 demonstrates that reversible access accommodates different selection rules. We adapt FastV-style importance (Chen et al., 2024) and PDrop-style stage schedules (Xing et al., 2025) to ReVisKV while preserving full visual KV during prefill. Both adaptations improve all three tasks at both budgets, with the largest gains on MMVet. On MIA-Bench, they improve scores by 8.1–13.2% relative to native pruning and preserve over 99.5% of the Dense score, extending the benefit to instruction following. Across the two selectors and budgets, they retain 97.8–99.0% of the Dense three-task average score.

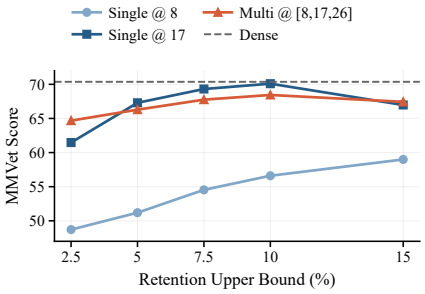

Figure 3: **MMVet quality across dynamic retention bounds and anchor layouts.**

4.4 Multi-anchor refresh

We vary anchor placement to test whether later refresh can preserve quality while offloading begins earlier. In Figure 3, the three-anchor layout improves MMVet over a single early anchor at every tested dynamic-budget configuration, with a 32.8% gain at the tightest bound. A single late anchor is competitive at intermediate bounds but keeps a longer prefix fully resident. Multiple anchors allow later layers to revise their visual focus without retaining that entire prefix on GPU.

The fixed-10% quality comparison in Table 4 reaches the same conclusion. The three-anchor layout obtains the highest sparse scores on both tasks. For the table's system columns, we separately measure the anchor layouts on 8K MMVet inputs with dynamic selection between 5% and 10% and 10% hot buffers. Three anchors require 31.8% less GPU visual-KV capacity than the single late anchor while using slightly less time per decoding step. Relative to the two-anchor layout [8, 17], adding the final anchor improves both task scores and reduces H2D traffic from 15.2 to 13.9 MB/token. Later reselection can therefore improve the working set without necessarily increasing recall traffic.

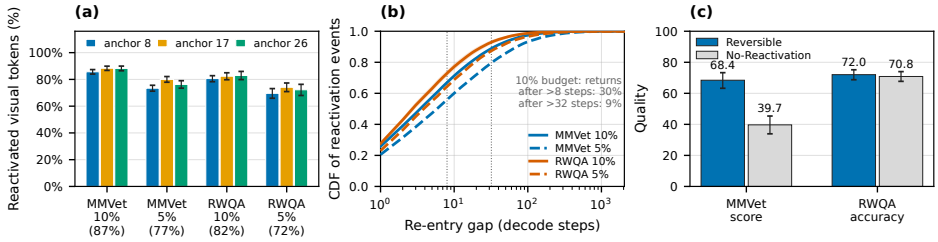

Figure 4: **Visual evidence returns after inactivity.** (a) Temporal reactivation rates; (b) distributions of re-entry gaps; (c) task quality with and without reactivation.

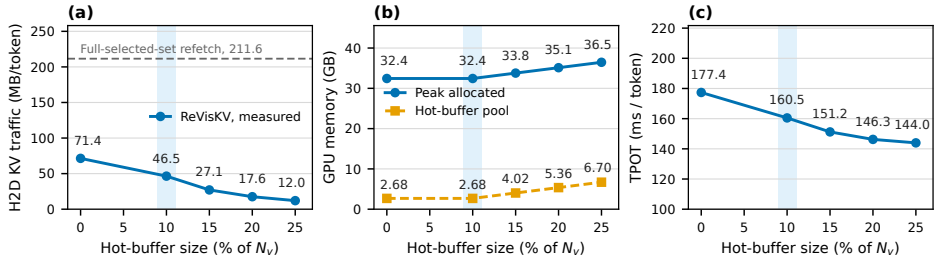

Figure 5: **Hot-buffer capacity trades memory for recall cost.** (a) H2D KV traffic; (b) GPU memory; (c) decode TPOT. The dashed line shows analytical full-selected-set refetch traffic.

4.5 Reactivation preserves access to visual evidence

To test whether inactive evidence is needed again, we track visual IDs that leave a working set and later return, then compare reversible selection with a policy that permanently bars returning IDs. Figure 4(a) shows that more than 80% of ever-selected IDs reactivate on both MMVet and RealWorldQA at the 10% upper bound. The re-entry gaps in Figure 4(b) include absences spanning multiple steps, showing that reuse extends beyond consecutive selections. The hot buffer serves returns whose KV remains resident, while CPU backing recovers entries evicted during longer absences.

Figure 4(c) shows a 42.0% MMVet score loss when reactivation is disabled. As reasoning proceeds, permanently barred entries cannot supply evidence needed again, making reversible access particularly valuable on this task. RealWorldQA changes much less, indicating that the impact depends on the task. The large VisionZip losses on InfoVQA and ChartQA in Table 1 likewise show that aggressive permanent reduction can be costly. Preserving a recovery path avoids committing all later reasoning to the evidence retained earlier.

4.6 Hot-buffer capacity controls recall cost

We vary hot-buffer capacity at a fixed selection policy to isolate the cost of retaining reusable KV. In the 32K-input workload of Figure 5, the 0% configuration keeps only the current working set, releasing entries as soon as they become inactive. It still needs a physical GPU pool large enough for the selection upper bound. The 10% configuration uses that same pool as a hot buffer, retaining inactive entries until their slots are needed. This reduces H2D traffic by 34.9% and TPOT by 9.5% with unchanged peak GPU memory and identical generated outputs.

Larger buffers retain more history, but each increase yields a smaller latency improvement. Expanding the buffer from 10% to 25% further reduces TPOT by 10.3% while adding 4.0 GB of peak GPU memory. The 10% default therefore captures a useful reuse benefit without enlarging the pool required for active KV, leaving more memory for additional requests. When GPU memory is available and transfer bandwidth is more constrained, a larger hot buffer offers an alternative that exchanges capacity for lower recall traffic.

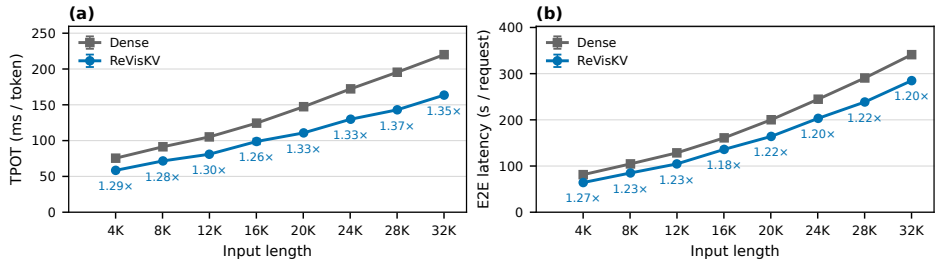

Figure 6: **Latency scaling with input length.** (a) Decode TPOT; (b) end-to-end model-inference latency.

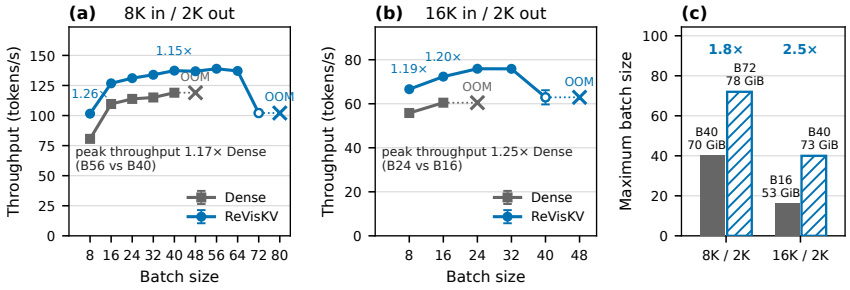

Figure 7: **Batch feasibility and throughput.** (a) Throughput at 8K input; (b) throughput at 16K input; (c) maximum feasible batch size. Hollow markers and hatched bars indicate high step-time variability ($p_{90}/\text{median} > 1.5$).

4.7 Latency and Batch Scalability

Figure 6 shows lower decode latency throughout the 4K–32K input range, with ReVisKV achieving 1.26 to 1.37× TPOT speedup at a matched batch size of eight and fixed output length. The benefit of sparse decoding therefore outweighs recall overhead in these workloads. End-to-end speedup is smaller, at 1.18 to 1.27×, because visual encoding and prefill still contribute to request latency. ReVisKV thus maintains a latency advantage across all evaluated input lengths, despite the increasing volume of visual context.

The batch-scaling results in Figure 7 show that reduced residency also expands the feasible batch range. ReVisKV supports 1.8× and 2.5× the maximum Dense batch size at 8K and 16K input, respectively. Peak observed throughput improves by 16.7% and 25.5% at these input lengths. The larger capacity gain at 16K is consistent with visual KV consuming more memory per request as context grows.

The throughput benefit also holds at matched batch sizes, improving by 15.2% to 26.0% at 8K and 19.4% to 19.6% at 16K. ReVisKV therefore supports both faster execution at the same batch size and larger feasible batches. Peak throughput occurs before the largest feasible batch, where step-time variability increases. We also adapt ReVisKV to the vLLM inference framework, with results reported in Appendix F.

5 Conclusion

ReVisKV reduces GPU visual-KV storage through complete CPU backing and small hot buffers in managed layers, preserving access to inactive evidence. Multi-anchor refresh adapts working sets across depth, while cross-step reuse and stage-shared execution limit recall cost. Across the evaluated workloads, this design preserves quality close to VisionPulse with substantially less GPU visual-KV capacity, while lowering latency and supporting larger batches than dense decoding.

AI USE STATEMENT

LLM assistants were used for literature retrieval, manuscript drafting and revision. MMVet and MIA-Bench use an LLM judge as part of their evaluation protocols. The authors are responsible for the final manuscript, experimental evidence, and disclosure of AI assistance.

REPRODUCIBILITY STATEMENT

The method specifies the selection rule, stage structure, and recall policy. The supplementary material records budget settings, evaluation protocols, and system workloads. Figure and table sources retain the underlying measurements and scoring records.

REFERENCES

Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. In *International Conference on Learning Representations*, pp. 86943–86967, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8be9c134bb193d8bd3827d4df8488228-Abstract-Conference.html.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025. URL https://arxiv.org/abs/2511.21631.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024. URL https://arxiv.org/abs/2406.02069.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *Computer Vision – ECCV 2024*, volume 15139 of *Lecture Notes in Computer Science*, pp. 19–35. Springer, 2024. doi: 10.1007/978-3-031-73004-7_2. URL https://doi.org/10.1007/978-3-031-73004-7_2.

Sixun Dong, Juhua Hu, Mian Zhang, Ming Yin, Yanjie Fu, and Qi Qian. MMTok: Multimodal coverage maximization for efficient inference of VLMs. In *International Conference on Learning Representations*, pp. 48075–48099, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4f8745c0ca5c951a92844ad7668cfca5-Abstract-Conference.html.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113152–113188, 2025. doi: 10.52202/085713-3773. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a40ff56daab9f4808b1e18350c8a11ce-Abstract-Conference.html.

Yihong Huang, Fei Ma, Yihua Shao, Jingcai Guo, Zitong Yu, Laizhong Cui, and Qi Tian. Nüwa: Mending the spatial integrity torn by VLM token pruning. In *International Conference on Learning Representations*, pp. 100825–100856, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a46156bd3579c3b268108ea6aca71d13-Abstract-Conference.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lee.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. InfographicVQA. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1697–1706, 2022. URL https://openaccess.thecvf.com/content/WACV2022/html/Mathew_InfographicVQA_WACV_2022_paper.html.

Yusu Qian, Hanrong Ye, Jean-Philippe Fauconnier, Peter Grasch, Yinfei Yang, and Zhe Gan. MIA-Bench: Towards better instruction following evaluation of multimodal LLMs. In *International Conference on Learning Representations*, pp. 35145–35165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/571082ea18d30060177dfcaf662ff0e5-Abstract-Conference.html.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. CAKE: Cascading and adaptive KV cache eviction with layer preferences. In *International Conference on Learning Representations*, pp. 89945–89977, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/dfae940651f3e690a12e19c874edad7c-Abstract-Conference.html.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 31094–31116. PMLR, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 57355–57373. PMLR, 2025. URL https://proceedings.mlr.press/v267/sun25b.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Lyuke Wang, Zhuo Li, and Guangxu Zhu. VisCache: Visual KV cache pruning for efficient vision large language model inference. *arXiv preprint arXiv:2608.24063*, 2026. URL https://arxiv.org/abs/2608.24063.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, Guanzhou Chen, Zichen Ding, Changyao Tian, Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Yuchen Duan, Xuehui Wang, Zhi Hou, Haoran Hao, Tianyi Zhang, Songze Li, Xiangyu Zhao, Haodong Duan, Nianchen Deng, Bin Fu, Yinan He, Yi Wang, Conghui He, Botian Shi, Junjun He, Yingtong Xiong, Han Lv, Lijun Wu, Wenqi Shao, Kaipeng Zhang, Huipeng Deng, Biqing Qi, Jiaye Ge, Qipeng Guo, Wenwei Zhang, Songyang Zhang, Maosong Cao, Junyao Lin, Kexian Tang, Jianfei Gao, Haian Huang, Yuzhe Gu, Chengqi Lyu, Huanze Tang, Rui Wang, Haijun Lv, Wanli Ouyang, Limin Wang, Min Dou, Xizhou Zhu, Tong Lu, Dahua Lin, Jifeng Dai, Weijie Su, Bowen Zhou, Kai Chen, Yu Qiao, Wenhai Wang, and Gen Luo. InternVL3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025. URL https://arxiv.org/abs/2508.18265.

xAI. RealWorldQA. Dataset introduced in Grok-1.5 Vision Preview, 2024. URL https://x.ai/news/grok-1.5v. Accessed 2026-09-24.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, pp. 21875–21895, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, and Dahua Lin. Conical visual concentration for efficient large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14593–14603, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Xing_Conical_Visual_Concentration_for_Efficient_Large_Vision-Language_Models_CVPR_2025_paper.html.

Hengbo Xu, Shengjie Jin, Yanbiao Ma, and Zhiwu Lu. VisionPulse: Dynamic visual sparsity for efficient multimodal reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.31457. arXiv:2605.31457.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. VisionZip: Longer is better but not necessary in vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19792–19802, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Yang_VisionZip_Longer_is_Better_but_Not_Necessary_in_Vision_Language_CVPR_2025_paper.html.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. MM-Vet: Evaluating large multimodal models for integrated capabilities. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 57730–57754. PMLR, 2024. URL https://proceedings.mlr.press/v235/yu24o.html.

Bowen Zeng, Feiyang Ren, Jun Zhang, Xiaoling Gu, Ke Chen, Lidan Shou, and Huan Li. HybridKV: Hybrid KV cache compression for efficient multimodal large language model inference. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13018–13034, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.594. URL https://aclanthology.org/2026.acl-long.594/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Algorithm 1 ReVisKV decoding with stage-shared incremental recall

Input: Prompt; anchors $a_1 < \cdots < a_M$; parameters $(\tau, r_{\min}, r_{\max})$; pool capacities $c_j \geq \lceil N_v r_{\max} \rceil$
Output: Generated tokens and updated per-layer KV caches

Run full-attention prefill; preserve each managed layer's visual KV in H_ℓ
Release those full GPU visual copies after host capture; retain text KV T_ℓ
Allocate layer-specific pools P_ℓ; initialize $C_{0,j}$, π_j, and LRU orders as empty
$y \leftarrow$ token sampled from the prefill output; emit y
for each decoding step t until the stopping condition **do**
    $h \leftarrow$ token embedding of y
    for each decoder layer ℓ in depth order **do**
        Form query q_ℓ and append the current token's KV using the backbone operations
        if ℓ is full-resident **then**
            Compute o_ℓ with full causal KV; retain probabilities A_ℓ at anchors
            if $\ell = a_j$ is an anchor **then**
                Select $I_{t,j}$ from all visual IDs using A_ℓ and Equations 1–2
            end if
        else
            $j \leftarrow$ stage containing managed layer ℓ
            if ℓ is the first managed layer of stage j **then**
                $\Delta I_{t,j} \leftarrow I_{t,j} \setminus C_{t-1,j}$
                Move hits $I_{t,j} \cap C_{t-1,j}$ to the MRU end in increasing ID order
                for each $i \in \Delta I_{t,j}$ in increasing ID order **do**
                    Assign a free slot, or replace the LRU entry outside $I_{t,j}$
                    Update π_j and mark i as MRU
                end for
                $C_{t,j} \leftarrow \mathrm{dom}(\pi_j)$
                Build shared codes for selected visual slots and causal text positions
                Enqueue pinned-memory uploads of codes and miss ID–slot pairs
            end if
            Enqueue RECALLMISSES$(H_\ell, P_\ell, \Delta I_{t,j}, \pi_j)$ if there are misses
            $D_\ell \leftarrow$ BUILDDESCRIPTORS(shared codes, P_ℓ, T_ℓ) on GPU
            $o_\ell \leftarrow$ SPARSEATTENTION(q_ℓ, D_ℓ) after recall and descriptor construction
        end if
        $h \leftarrow$ FINISHLAYER(h, o_ℓ) using the backbone's remaining layer operations
    end for
    $y \leftarrow$ token sampled from the final hidden state; emit y
end for

A IMPLEMENTATION AND MEMORY SEMANTICS

Algorithm 1 describes the default cached-reuse path for one request, and the runtime batches these operations across active requests. For each managed layer ℓ, H_ℓ stores the original-precision visual KV in pinned CPU memory, P_ℓ is its preallocated GPU hot pool, and T_ℓ holds prompt-text and generated-token KV. Each stage j maintains a resident-ID set $C_{t,j}$, an ID-to-slot map π_j, and an LRU order. The pool capacity c_j accommodates the selection upper bound. Full-resident layers comprise the prefix and anchors defined in Section 3.

At the first managed layer of a stage, selected IDs are processed in increasing order, matching the implementation's cache updates. Selected resident entries are moved to the most-recently-used end before misses are assigned slots, so replacement never evicts a currently selected ID. The code table includes only selected visual slots and causal text positions, while cached visual history remains available for reuse without participating in attention. Identical initial states, selections, capacities, and replacement rules allow the slot map to be shared across the stage, while KV values remain layer-specific.

RECALLMISSES and BUILDDESCRIPTORS correspond to the two Triton kernels. The former gathers missing KV directly from mapped pinned CPU memory, whereas the latter resolves shared codes against each layer's pool and text-KV addresses. Metadata uploads, recall, descriptor construction, and attention are enqueued in stream order, ensuring that every read follows its required writes. Asynchronous submission lets the CPU prepare subsequent work without a host-side wait at each managed layer.

Table 5: **Dynamic retention bounds and budget temperatures.** Bounds are percentages of the input visual tokens.

Upper bound (%)	2.5	5	7.5	10	15
τ	0.1	0.1	0.4	0.4	0.4
Lower bound (%)	1	1	5	5	5

Table 6: **Quality across dynamic retention bounds and anchor layouts.** Columns give the retention upper bound. Scores are on a 0–100 scale.

Dataset	Anchors	2.5%	5%	7.5%	10%	15%
MMVet	[8]	48.7156	51.1927	54.5413	56.6055	58.9908
	[17]	61.4679	67.2936	69.3119	70.0917	66.9725
	[8, 17, 26]	64.6789	66.2844	67.7523	68.4404	67.4312
RealWorldQA	[8]	71.1111	71.5033	72.4183	73.2026	71.2418
	[17]	72.0261	73.0719	72.9412	72.4183	71.8954
	[8, 17, 26]	72.4183	71.5033	72.1569	72.0261	72.1569

GPU residency measures allocated visual-KV capacity, including full-resident layers and managed-layer hot buffers, relative to the dense visual cache. It does not measure how many entries are attended to or how many buffer slots are valid at a particular step. Peak GPU memory additionally includes model weights, text and generated-token KV, and temporary workspaces.

B Evaluation protocol

We evaluate the complete benchmark splits, using the validation split for InfoVQA and the test split for ChartQA. Qwen models use BF16, temperature 0.7, top-p 0.8, and top-k 20. Generation ends at EOS or the 8192-token limit, and capped outputs remain in the evaluation. InternVL uses its thinking-mode system prompt. Model, preprocessing, prompt, generation settings, and scorer are held fixed within each backbone comparison.

MMVet and MIA-Bench use their task-specific judge rubrics with `gpt-4o-mini-2024-07-18`. RealWorldQA uses the VLMEvalKit answer extractor followed by exact matching, InfoVQA uses ANLS, and ChartQA uses relaxed accuracy.

Output lengths include reasoning and generated terminal tokens. Relative changes are computed against the matching Dense configuration from unrounded values. Avg gives equal weight to the reported datasets, averaging lengths and scores separately where both are present. Score retention expresses the score as a percentage of the stated baseline, with cross-dataset comparisons using these unweighted mean scores. All quality scores use a 0–100 scale. The 10% and 5% budgets are dynamic upper bounds for VisionPulse and ReVisKV, whereas native pruning retains a fixed subset, with PDrop's budget denoting final-stage retention.

C Dynamic-budget configurations and additional results

Each column of Table 5 defines a dynamic-budget configuration shared by all anchor layouts. The temperature τ controls visual-mass estimation in Equation 2 and is distinct from the generation temperature. Because the lower bound and temperature also vary across columns, the sweep evaluates complete configurations rather than the upper bound alone.

The additional RealWorldQA results in Table 6 show smaller, nonmonotonic differences across layouts and budgets than those observed on MMVet. In contrast to this dynamic sweep, the fixed-retention anchor study holds the selected fraction at 10%. Scores from the two studies therefore evaluate different selection policies.

Table 7: **System costs across anchor layouts.** Paired values report MMVet / RealWorldQA.

Anchors	H2D (MB/token)	TPOT (ms)	Peak HBM (GB)
[8]	8.4 / 7.4	55.4 / 55.3	15.3 / 15.4
[17]	11.9 / 8.8	70.1 / 69.3	17.5 / 17.5
[8, 17]	15.2 / 12.5	62.6 / 62.2	15.6 / 15.6
[17, 26]	11.4 / 9.2	75.9 / 75.4	17.8 / 17.8
[8, 17, 26]	13.9 / 13.2	68.4 / 68.6	15.8 / 15.8

D TRACE ANALYSIS AND REACTIVATION

The trace analysis replays recorded generations so that different layers are examined along the same output sequence. Selected IDs are mapped to the original visual-token positions. Cross-layer similarity uses Jaccard overlap in the heatmap and overlap normalized by the target set size in the layer-distance curve. Temporal reuse is the fraction of the later working set already present at the earlier step. Layer-distance and step-gap curves weight MMVet and RealWorldQA equally. The adjacent-step cumulative distributions pool anchor-step observations within each dataset, giving equal weight to each event.

A reactivation occurs when an ID is selected, subsequently remains absent for at least one complete decoding step, and is selected again at the same anchor. The reactivation rate measures the proportion of ever-selected IDs that return, and re-entry-gap distributions pool return events. The quality ablation permanently bars an ID after an active-set exit. Once barred, an ID cannot fill a later selection, so the ablation also reduces the usable working set over time. It measures the combined consequence of permanent exclusion, rather than isolating reactivation at a matched active-set size.

E SYSTEM WORKLOADS AND SUPPLEMENTARY MEASUREMENTS

System experiments run in BF16 on an A800-80GB. Dense and ReVisKV share normalization, rotary-position, packing, and padding optimizations, and ReVisKV additionally uses stage-shared recall and an attention CUDA graph. Latency and hot-buffer sweeps use matched RealWorldQA inputs at batch size eight and generate 1024 tokens with EOS disabled. The batch-scaling study uses 8K and 16K input buckets, grouped prefill, and 2048 output tokens per request.

TPOT measures decode time per sequential step, while aggregate throughput counts generated tokens across the batch per unit time. End-to-end latency includes visual encoding, language-model prefill, and decode, excluding input loading and preprocessing. H2D reports visual-KV payload per generated token in decimal MB. Memory values in Figure 5 and Table 7 use decimal GB, while the annotations in Figure 7 use GiB.

Table 7 reports anchor-system measurements using dynamic selection with a 10% upper bound, a 10% hot buffer, and 8K inputs. The main anchor table uses MMVet H2D from this workload alongside separate fixed-10% quality results. The paired values here also report RealWorldQA.

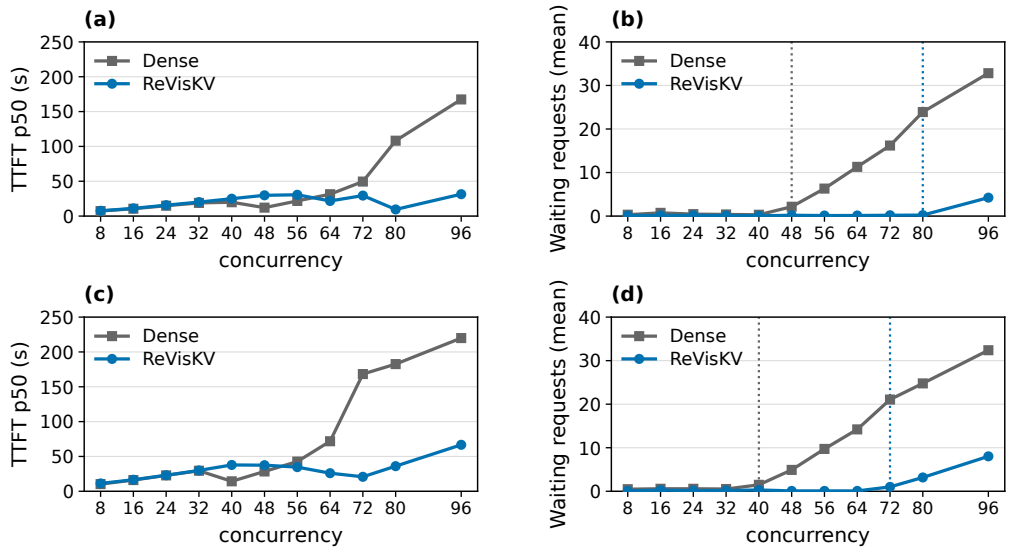

Figure 8: **Serving behavior of Qwen3-VL-Thinking in vLLM.** (a) Median TTFT for 4B; (b) mean waiting queue for 4B; (c) median TTFT for 8B; (d) mean waiting queue for 8B. Dotted lines mark the first observed concurrency with preemptions.

F Integration with vLLM serving

We integrate ReVisKV into vLLM (Kwon et al., 2023) through a custom attention backend and KV-cache manager. The backend preserves full-context anchor selection, while managed layers attend through vLLM's paged KV layout. At the first decoding step, the manager copies managed-layer visual KV to pinned host memory and returns visual blocks outside the hot buffer to the shared GPU block pool. Text KV and boundary blocks remain resident. Within each stage, GPU lookup and replacement tables encode shared cache decisions. Triton kernels gather misses into paged hot-buffer slots on a copy stream, with per-layer events enforcing completion before attention. This integrates reversible residency with continuous batching, request completion, and preemption.

The serving evaluation uses Qwen3-VL-Thinking 4B and 8B on an A800-80GB with vLLM 0.29.0. With EOS disabled, requests contain approximately 8K visual tokens and generate 2048 tokens across closed-loop concurrencies from 8 to 96. Both systems use the same scheduling settings with prefix caching disabled. ReVisKV uses anchors $[8, 17, 26]$, dynamic selection bounded between 5% and 10%, and a block-aligned 10% hot buffer. The final managed layer also keeps its visual KV resident while retaining sparse attention, balancing vLLM's cache groups into 12 full-resident and 24 offloaded layers. Figure 8 reports client-observed time to first token (TTFT), including admission delay and prefill, and the server's mean waiting queue.

Figure 8(b) and (d) show that ReVisKV delays sustained queue growth as concurrency increases. Compared with Dense, the concurrency at which preemption is first observed increases by 66.7% for 4B and 80.0% for 8B. At concurrency 96, the mean waiting queue is reduced by 87.1% and 75.2%, respectively. The latency curves in Figure 8(a) and (c) reflect this admission benefit, with median TTFT reduced by 81.2% for 4B and 69.7% for 8B relative to Dense. At low concurrency, TTFT is similar to or slightly higher than that of Dense, so the advantage emerges under memory pressure rather than from faster prefill. These results show that reversible visual-KV residency can improve responsiveness within a complete serving engine by making GPU cache capacity available to additional requests.