PACE-VL: PRESERVING ATTENTION VIA CLUSTERED EVICTIONS FOR VISION–LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models that ingest high-resolution images or multi-page documents produce long, predominantly visual key–value (KV) caches, making decode-time memory and latency primary bottlenecks. Under a strict cache budget, the central challenge is preserving task quality when the vast majority of physical slots must be aggressively evicted. We introduce PACE-VL, a training-free KV-cache compression method that jointly designs retention, residual aggregation, and decode-time reading. Specifically, query states from the question or instruction tokens score the history to retain the highest-scoring entries, called evidence tokens, with their original keys and values. The remaining historical entries are clustered into anchors, cache entries that each store a selected original key, an aggregated value sum, and a multiplicity count—the number of tokens represented. At decode time, attention incorporates cluster multiplicities into normalization, so each cluster maintains its aggregate contribution under a shared-anchor-key approximation. Evidence tokens, anchors, and a short, uncompressed recent window occupy a cache of exactly the budgeted size. Across five visual-heavy benchmarks on three open VLMs spanning the Qwen and Llama-3 lineages, a single fixed configuration of PACE-VL without any fine-tuning outperforms all compared compression baselines in all 21 evaluated settings at a 10% cache budget. It matches or exceeds full-cache decoding in 11 settings and otherwise trails it by at most 1.87 points.

1 INTRODUCTION

Vision-language models (VLMs) increasingly operate on inputs dominated by visual tokens: high-resolution images, multi-page documents, OCR-heavy scenes, and long interleaved image–text contexts. KV-cache compression is attractive here because it acts after the model has encoded the prompt and targets the memory that decoding must touch, yet aggressive pruning can remove the evidence that fine-grained recognition or spatial reasoning needs. Existing compressors score the cached entries and keep a budgeted subset (Zhang et al., 2023; Xiao et al., 2024; Li et al., 2024; Cai et al., 2025; Tu et al., 2025); some fold discarded entries into survivors rather than deleting them (Zhang et al., 2024; Wan et al., 2024). How those survivors are then read at decode time receives less design attention than the eviction step itself: once the cache is compressed, ordinary attention still treats each remaining entry as a single key–value pair.

Two attention patterns motivate joint selection and aggregation. Across document and long-context settings on Qwen3-VL-8B, the top 1% of history entries ranked by decode attention capture 76–81% of its mass, and the top 10% capture 93–94% (§4.3). This concentration favors retaining important entries individually as *evidence tokens*, preserving their original keys and values. Yet a 10%-budget keep set selected by the retention rule below leaves 4–8% of full-cache decode attention outside the cache; deleting these entries inflates surviving weights by 5–11%. The remaining history therefore also needs an aggregate representation. We call historical tokens not retained as evidence the *residual history*. The challenge is to represent it through task-relevant *anchors*: cache entries that each store a selected original key, an aggregated value sum, and the number of tokens it represents (its *multiplicity*), enabling decoding over a smaller cache.

We therefore propose PACE-VL (Preserving Attention via Clustered Evictions for Vision–Language Models), a training-free KV-cache compression method. *Task-field-guided evidence retention* (§3.2)

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.

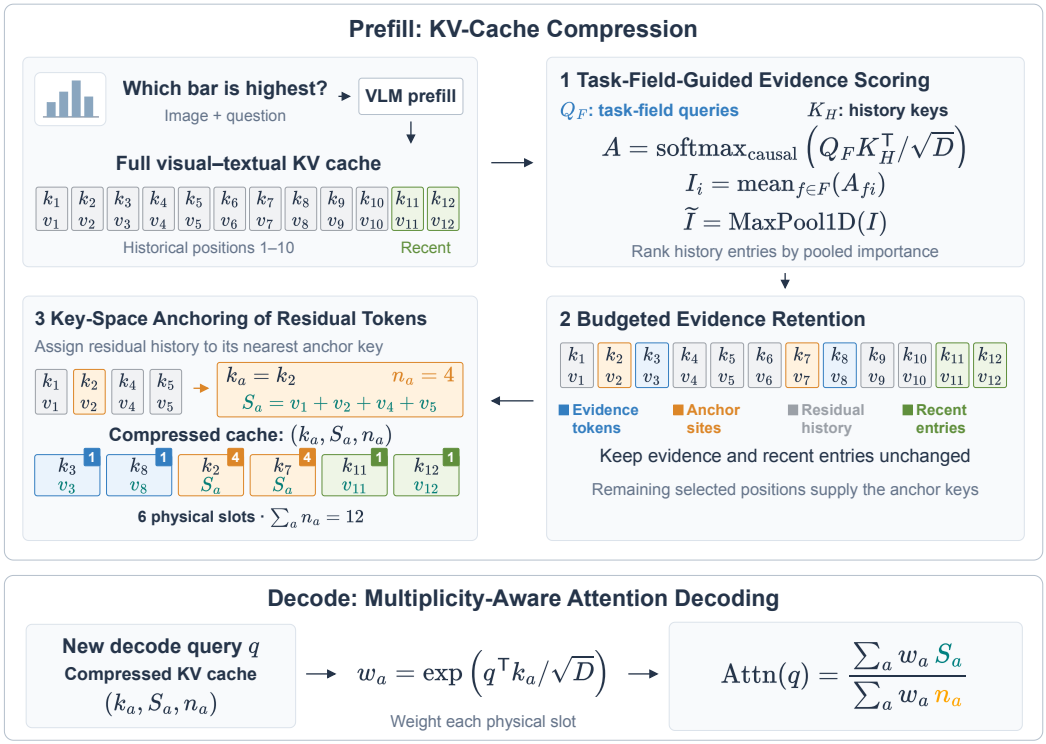


Figure 1: **PACE-VL overview. Prefill (top):** Task-field scoring selects evidence and anchor sites; residual tokens are aggregated by nearest key. Evidence and recent entries stay intact. **Decode (bottom):** Anchors' value sums and counts enter the attention numerator and denominator, respectively, recovering full attention under shared anchor keys. The example maps 12 tokens to 6 slots; selections are illustrative and colors denote cache roles.

uses queries from the *task field*—the question or instruction text—to rank the history and retain high-scoring entries individually under an exact cache budget. *Key-Space Anchoring of Residual Tokens* (§3.3) selects *anchor sites*, the original token positions supplying the anchor keys, from the highest-scoring remaining positions. It assigns residual tokens to their nearest anchor key and stores each group's value sum and multiplicity. *Multiplicity-Aware Attention Decoding* (§3.4) uses these sums in the numerator and multiplicities in the normalization to account for the tokens each anchor represents. Evidence tokens, anchors, and a short, uncompressed window of the most recent tokens share one fixed physical-slot budget. Decoding reads this smaller cache to reduce KV-cache traffic and accelerate generation. Figure 1 maps these modules to the prefill and decode stages.

We evaluate the complete PACE-VL pipeline at a 10% cache budget on five visual-heavy benchmarks—ChartQA, DocVQA, InfographicVQA, TextVQA, and MMLongBench at three context scales—across three open VLMs spanning the Qwen and Llama-3 lineages. Under this budget, PACE-VL is the strongest among the compressed methods we compare in all 21 benchmark–model–scale settings, matches or exceeds full-cache decoding in 11, and otherwise trails it by at most 1.87 points; no competing compressed baseline exceeds the full cache in any of the 21. At a 50% budget (§4.2), PACE-VL, HybridKV, and VL-Cache all remain within 0.62 points of full-cache performance on the short-form benchmarks.

Our contributions are threefold.

Task-field-guided evidence retention under an exact cache budget. We use queries from the located task field to score historical KV entries, then allocate a strict cache budget to exact evidence tokens, residual anchors, and recent entries (§3.2). This training-free construction preserves high-scoring evidence verbatim and occupies exactly the budgeted number of physical slots, while every original prompt token is represented once, individually or through an anchor.

Anchored residual aggregation with multiplicity-aware decoding. We aggregate residual tokens around task-selected anchor keys, storing value sums and multiplicity counts (§3.3). Decoding incorporates these counts through size correction (Bolya et al., 2023) to account for the population represented by each anchor (§3.4). The construction recovers full-cache attention when all keys within each cluster equal its anchor key. Otherwise, key-space distance bounds each token's logit perturbation, and nearest-anchor assignment minimizes this bound for fixed anchors. At an equal physical budget, spending those slots on additional verbatim entries instead of aggregating the residual is the costliest of the four component controls we test (§4.3).

Strict-budget empirical superiority and component ablations. Experiments span five benchmarks and three VLMs. At a 10% budget, the four matched-budget component controls reduce macro-average scores by 0.59–1.16 points across 20 MMLongBench datasets, with the largest effect on image counting (§4.3). We also measure generation latency and peak memory (§4.4).

2 Related Work

Visual Token Reduction. Visual-token reduction shortens the sequence processed by subsequent network layers. FastV uses attention to prune visual tokens in decoder layers (Chen et al., 2024). SparseVLM uses selected text tokens to assess visual relevance, adapts sparsification across layers, and recycles part of the pruned tokens into compact representations (Zhang et al., 2025). In vision transformers, ToMe combines similar tokens through bipartite matching (Bolya et al., 2023). These methods modify the token sequence within the network. PACE-VL instead compresses each layer's prefill KV cache for subsequent decoding, retaining selected entries verbatim and representing the residual history through anchors under a fixed physical-slot budget.

KV-Cache Compression for Vision–Language Models. Multimodal cache methods allocate limited slots across visual and textual context. VL-Cache uses modality-aware scoring and layer-adaptive budgets (Tu et al., 2025); HybridKV combines text-centric head classification with hierarchical budgets and head-specific pruning or retrieval (Zeng et al., 2026). AirCache estimates inter-modal relevance through an elite observation window (Huang et al., 2025), while MixKV balances importance and diversity using head-wise redundancy (Liu et al., 2026). LOOK-M combines text-prior retention with multimodal KV merging (Wan et al., 2024). These methods build on recency, attention-based selection, and layer- or head-dependent policies (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2023; Cai et al., 2025; Ge et al., 2024). For scoring queries, SnapKV uses a fixed prompt suffix (Li et al., 2024), whereas PACE-VL uses task-field queries to allocate exact evidence tokens and original-key residual anchors under one exact prompt-cache budget.

Token Aggregation and Attention Normalization. Cache aggregation includes CaM's attention-guided merging into retained entries (Zhang et al., 2024), KVMerger's Gaussian-weighted key-similarity groups (Wang et al., 2024), and WeightedKV's attention-weighted value combinations under retained keys (Yuan et al., 2025). Unlike CaM and LOOK-M, which merge unselected entries without multiplicity adjustments, PACE-VL tracks cluster counts and incorporates them into decode-time normalization. Size-aware reading also has precedents: ToMe adds $\log s$ to attention logits for merged tokens (Bolya et al., 2023). In KV decoding, RazorAttention preserves full caches for retrieval heads and summarizes discarded entries in other heads with a mean-key, mean-value compensation token, weighted by the number of entries it represents (Tang et al., 2024). PACE-VL instead partitions history into evidence and anchor entries within each head's fixed budget. Attention Matching fits per-entry logit biases and values using reference queries to match attention outputs and unnormalized attention mass (Zweiger et al., 2026), whereas PACE-VL constructs anchor values and weights directly as sums and integer counts, without reference-query fitting.

3 PACE-VL

We present PACE-VL in two parts. Section 3.2 introduces task-field-guided evidence retention: task-field queries score the history, and high-scoring entries are retained individually under an exact cache budget. Sections 3.3–3.4 introduce anchored residual aggregation and multiplicity-aware decoding: residual tokens are grouped around anchor keys, and their value sums and counts are used during

decoding to approximate their aggregate attention contribution. Algorithm 1 summarizes prefill compression and decoding.

3.1 PRELIMINARIES

A transformer-based VLM receives a prompt of T visual and textual tokens and stores their keys and values after prefill (Vaswani et al., 2017). PACE-VL operates layer by layer on this cache. At one layer, let the prefill states be queries $Q \in \mathbb{R}^{B\times H_q\times T\times D}$ and keys and values $K, V \in \mathbb{R}^{B\times H_{kv}\times T\times D}$, where B is the batch size, H_q and H_{kv} are the query and KV head counts, and D is the head dimension; queries and keys are taken after rotary positional encoding (RoPE). Under grouped-query attention, the query heads that share a KV head are averaged before scoring; scoring and compression below are per KV head. Let R denote the protected recent window; positions $1 \le i \le T - R$ form the *history*, the only part of the cache that is compressed. Suppressing batch and head indices, a decode query q attends over the full cache as

$$\mathcal{A}^\star(q) \;=\; \frac{\sum_{i=1}^{T} \exp\!\big(q^\top k_i/\sqrt{D}\big)\, v_i}{\sum_{i=1}^{T} \exp\!\big(q^\top k_i/\sqrt{D}\big)}. \tag{1}$$

PACE-VL maps the prefill cache $(K, V)_{1:T}$ to a compressed cache $\hat{\mathcal{C}}$ of C physical entries, with a headline budget of $C/T \approx 0.1$, and designs decoding to approximate Eq. (1).

Within this cache, evidence tokens retain their original keys and values; the other historical tokens form the residual history. Each anchor is a cache entry (k_a, S_a, n_a) representing a group of residual tokens: k_a is the original key at its selected anchor site, S_a is the group's value sum, and n_a is its multiplicity. The recent window is retained separately.

For a decode query q, define the unnormalized *attention mass* of a subset S of cached positions as $\mathcal{M}_q(S) = \sum_{i\in S} \exp(q^\top k_i/\sqrt{D})$. A selection-only scheme retains a set $\mathcal{R}$ with $|\mathcal{R}| = C$ and evicts its complement $\mathcal{E}$. Recomputing Eq. (1) over $\mathcal{R}$ sets evicted logits to $-\infty$: it removes $\mathcal{M}_q(\mathcal{E})$ from the denominator and loses the evicted values. PACE-VL instead represents this population through shared anchor keys, with bounded per-token logit error (§3.4).

3.2 TASK-FIELD-GUIDED EVIDENCE RETENTION UNDER AN EXACT CACHE BUDGET

Visual relevance depends on the question: the same chart may be queried about a bar, its axis units, or its legend. Relevance in a multimodal prompt is a property of the image–question pair rather than of the visual history alone, which is why PACE-VL derives its score from the question itself. A benchmark or runtime adapter locates the question or instruction tokens at positions $F = \{f_1, \dots, f_m\}$ in the full multimodal prompt. Every resolved position supplies a query probe, avoiding the truncation or dilution that a fixed trailing window can introduce. These textual queries score both visual and textual keys, ranking the whole history by task relevance.

Each history position $1 \le i \le T - R$ receives the causal attention mass it draws from the task-field probes, averaged over probes. Here q_f denotes the group-averaged scoring query for a KV head:

$$I_i \;=\; \frac{1}{|F|} \sum_{f\in F} \frac{\mathbf{1}[\,i \le f\,]\, \exp\!\big(q_f^\top k_i/\sqrt{D}\big)}{\sum_{j=1}^{\min(f,\,T-R)} \exp\!\big(q_f^\top k_j/\sqrt{D}\big)}. \tag{2}$$

As a mean of causal attention distributions, I is nonnegative and sums to one regardless of $|F|$ and T. The implementation evaluates Eq. (2) with an exact two-pass streaming softmax (§3.5) and then smooths task-field importance along the token axis with same-length max pooling of kernel size k, $\tilde{I} = \mathrm{MaxPool1D}(I, k)$, following SnapKV's pooling practice for retained-position selection (Li et al., 2024). Pooling extends high scores to neighboring sequence positions, favoring retention of locally informative stretches. Pooling is a maximum, so $\tilde{I}$ is not normalized: each $\tilde{I}_i$ is the largest probe-averaged attention probability within radius $\lfloor k/2 \rfloor$ of position i. The fixed threshold τ used below is defined on this scale; Eq. (3) bounds the evidence-token count.

One condition constrains the construction: attention is causal, so a probe at position f scores only history at $i \le f$. The multimodal layouts we study put the visual evidence before the question,

which is what makes the scoring well posed here; a question-first layout would leave the subsequent evidence unranked and need a different probe construction (Appendix A.9).

We use these pooled scores to divide the cache budget and select evidence tokens. Given a cache ratio r, the recent window R, and a minimum cache size $C_{\min}$, the compressed prompt-cache size is $C = \min\big(T, \max(C_{\min}, R+1, \lfloor rT \rfloor)\big)$. Whenever compression is active, the compressed cache contains *exactly* C physical entries. The recent window preserves R entries for decoding continuity, leaving $C - R$ slots for evidence tokens and anchors. The evidence-token count adapts to the pooled task-field importance: writing $P = \mathrm{round}\big(\mathrm{mean}_{b,h} |\{\, i : \tilde{I}_i \geq \tau \,\}|\big)$ for the threshold pass count, averaged over the batch and KV heads and rounded to an integer, and with clamp fractions $0 \leq \alpha_{\min} \leq \alpha_{\max} < 1$,

$$\begin{aligned} N_{\mathrm{evid}} &= \mathrm{clamp}\Big(P,\ \big\lfloor \alpha_{\min}(C-R) \big\rfloor,\ \min\big(\big\lfloor \alpha_{\max}(C-R) \big\rfloor,\ C-R-1\big)\Big), \\ N_{\mathrm{anchor}} &= C - R - N_{\mathrm{evid}}, \end{aligned} \tag{3}$$

The allocation reserves at least one anchor to represent the residual history. The budget therefore divides into three parts with distinct roles: R recent entries held for decoding continuity, N_{evid} individually addressable evidence tokens, and N_{anchor} anchors. The top $C - R$ history positions by smoothed task-field importance are selected, and the first N_{evid} of them become evidence tokens, carried into the compressed cache with their keys and values unchanged.

3.3 Key-Space Anchoring of Residual Tokens

Key-space anchoring constructs these representative entries by grouping residual tokens according to their key vectors. The remaining N_{anchor} selected positions become anchor sites and supply the original keys k_a. Each residual history token is assigned to its nearest anchor $a(i)$ by squared Euclidean distance between post-RoPE keys; each anchor owns its source token, so no anchor is empty. Let $\mathcal{A}_a$ be the set of tokens assigned to anchor a. The anchor stores their value sum $S_a = \sum_{i \in \mathcal{A}_a} v_i$ and multiplicity $n_a = |\mathcal{A}_a| \geq 1$, preserving their aggregate value content and count in one physical entry. The final physical cache is ordered as [evidence tokens | anchors | recent entries], with evidence tokens and recent entries carrying multiplicity 1 and their original values.

Proposition 1 (Exact budget and complete population coverage)**.** *Whenever compression is active, the physical cache contains exactly C entries, and the multiplicities satisfy $\sum_a n_a = T$. Complete coverage requires $N_{\mathrm{anchor}} \geq 1$, which Eq. (3) enforces.*

The proof is in Appendix A.6. Two properties of the grouping deserve emphasis. An anchor stores the key of an *actually selected* token rather than a cluster centroid, preserving an observed post-RoPE key; a centroid need not coincide with any original cache key. Anchor sites are the highest-scoring positions left after the evidence-token slots are filled, rather than random ones, so that anchor keys lie where the task field attends. Replacing either choice—centroid keys, or randomly drawn anchor sites—lowers the macro-average in §4.3. Distances use post-RoPE keys, which encode position.

Reading this compressed cache requires a decode rule that uses both the aggregated values and the number of tokens each slot represents.

3.4 Multiplicity-Aware Attention Decoding

An anchor occupies one physical slot but stands for n_a original tokens. Ordinary attention applied to (k_a, S_a) would retain the group's value sum in the numerator but count its key only once in the denominator, omitting the represented multiplicity. A query q therefore attends over the compressed cache with the anchor populations restored in the normalization:

$$\mathrm{Attn}(q) \;=\; \frac{\sum_a \exp\big(q^\top k_a/\sqrt{D}\big)\, S_a}{\sum_a \exp\big(q^\top k_a/\sqrt{D}\big)\, n_a}, \tag{4}$$

where a ranges over all physical entries. The numerator weights each entry by the sum of the values it absorbed and the denominator by how many tokens it represents; for evidence tokens and recent entries ($n_a = 1$, $S_a = v_a$) the expression reduces to standard attention. It is the KV-cache analogue of ToMe's size-weighted correction (Bolya et al., 2023) with n_a in the role of the merged size s (§2).

Algorithm 1 PACE-VL: Prefill Compression and Multiplicity-Aware Decoding

Require: Prefill K, V and task-field queries $\{q_f\}_{f \in F}$, with batch/head indices suppressed
Require: $r, C_{\min}, R, \tau, \alpha_{\min}, \alpha_{\max}, k$, with $0 \le \alpha_{\min} \le \alpha_{\max} < 1$
Scoring queries q_f are averaged within each GQA group (§3.1).

Prefill Phase
$C \leftarrow \min(T, \max(C_{\min}, R+1, \lfloor rT \rfloor))$ ▷ whole-prompt budget, §3.2
if $C = T$ **then**
  return $\hat{\mathcal{C}} = [(k_i, v_i, 1)]_{i=1}^{T}$ ▷ no compression
end if
Compute I_i by Eq. (2); $\tilde{I} \leftarrow \mathrm{MaxPool1D}(I, k)$ ▷ mean causal task-field attention
$P \leftarrow \mathrm{round}\Big(\mathrm{mean}_{b,h} \sum_{i=1}^{T-R} \mathbf{1}[\tilde{I}_{bhi} \ge \tau]\Big)$ ▷ batch/KV-head mean
Compute $N_{\mathrm{evid}}, N_{\mathrm{anchor}}$ from P by Eq. (3) ▷ $N_{\mathrm{anchor}} \ge 1$
Rank the top $C - R$ history positions by $\tilde{I}$; retain the first N_{evid} as evidence
Use the remaining N_{anchor} positions as anchor sites, supplying keys k_a
$S_a \leftarrow \mathbf{0}$; $n_a \leftarrow 0$ for each anchor a
for each residual history token i **do** ▷ evidence and recent entries excluded
  $a(i) \leftarrow \arg\min_a \|k_i - k_a\|_2^2$ ▷ anchor sites own themselves on ties
  $S_{a(i)} \leftarrow S_{a(i)} + v_i$; $n_{a(i)} \leftarrow n_{a(i)} + 1$
end for
Assemble $\hat{\mathcal{C}}$ as [evidence | anchors | recent] ▷ exact entries store $(k_i, v_i, 1)$
return $\hat{\mathcal{C}}$ ▷ C slots; $\sum_a n_a = T$

Decode Step (inputs: cache $\hat{\mathcal{C}}$ and current $q, k_{\mathrm{new}}, v_{\mathrm{new}} \in \mathbb{R}^D$)
Append $(k_{\mathrm{new}}, v_{\mathrm{new}}, 1)$ to $\hat{\mathcal{C}}$ ▷ new token has multiplicity 1
$\mathrm{Attn}(q) \leftarrow \dfrac{\sum_a \exp(q^\top k_a/\sqrt{D})\, S_a}{\sum_a \exp(q^\top k_a/\sqrt{D})\, n_a}$ ▷ all physical entries; Eq. (4)
return $\mathrm{Attn}(q)$ and updated cache $\hat{\mathcal{C}}$

Writing $\bar{v}_a = S_a/n_a$ for the anchor's mean value, Eq. (4) is algebraically identical to an ordinary softmax over logits shifted by $\log n_a$, applied to those means. This form permits fused evaluation with an attention kernel that supports the additive $\log n_a$ bias.

The budget C applies to the compressed prompt cache. During generation, each new key k_{new} and value v_{new} are appended unchanged with multiplicity 1. After g generated tokens have been cached, the physical cache has $C + g$ entries and represents $T + g$ tokens; Eq. (4) applies to all of these entries.

Proposition 2 (Exactness under shared anchor keys)**.** *If $k_i = k_a$ for every $i \in \mathcal{A}_a$ and every anchor a, then Eq. (4) equals the full attention $\mathcal{A}^\star(q)$ of Eq. (1) over all T original tokens, for every query q.*

Grouping the sums of Eq. (1) by the parts $\mathcal{A}_a$, which partition $\{1, \dots, T\}$ by Proposition 1, and substituting the shared key gives Eq. (4) term by term (Appendix A.6). Anchor keys are not shared exactly, so the useful statement is how the error behaves when they are not: let $\delta_i = q^\top \big(k_{a(i)} - k_i\big)/\sqrt{D}$ denote the logit perturbation.

Proposition 3 (Bounded perturbation and the clustering objective)**.** *Eq. (4) equals the exact attention of Eq. (1) computed with each logit $q^\top k_i/\sqrt{D}$ replaced by $q^\top k_i/\sqrt{D} + \delta_i$, and $|\delta_i| \le \|q\|\, \|k_{a(i)} - k_i\|/\sqrt{D}$. Consequently, among all assignments of history tokens to a fixed set of anchor sites, assigning each token to its nearest anchor in key space minimizes this bound token-wise.*

The bound is Cauchy–Schwarz, and $\|q\|/\sqrt{D}$ does not depend on the assignment (Appendix A.6); the nearest-anchor assignment in Algorithm 1 therefore minimizes this bound for fixed anchors. In these terms a selection-only scheme is the special case $\delta_i = -\infty$ for every evicted token (§3.1); PACE-VL keeps the identical budget and replaces that unbounded perturbation with one bounded by each cluster's key-space radius. The bound grows with each token's distance to its assigned anchor; exactly retained evidence tokens incur no key-replacement error.

3.5 Implementation and Computational Cost

Scoring and cache construction are performed once per layer after prefill; subsequent decoding reads the compressed cache. Eq. (2) takes two streaming passes over the history, $O(|F|\,T\,D)$ time per KV head with memory bounded by the chunk size rather than by $|F| \times T$; anchor assignment costs $O(T\,N_{\text{anchor}}\,D)$, quadratic in the prompt length at $N_{\text{anchor}} = O(C)$ and $C = rT$, and is incurred once during cache construction. With g generated tokens cached, decoding attends over $C+g$ entries instead of $T + g$. Attention work and key/value storage scale with these lengths, giving a ratio of $(C + g)/(T + g)$, with an additional scalar multiplicity per compressed-cache entry.

The same budget applies to every layer and head, keeping the compressed cache in a dense tensor of uniform shape for indexing and batching. The parameters are the cache ratio r, recent window R, minimum cache size $C_{\min}$, pooling kernel k, task-field importance threshold τ, and clamp fractions $(\alpha_{\min}, \alpha_{\max})$; all experiments use one fixed configuration, named in Appendix A.7.

4 Experiments

We evaluate the quality of the complete PACE-VL pipeline under a strict cache budget, examine the attention patterns that motivate its construction, and measure its inference costs. The main comparison uses a 10% budget; the 50% results provide a moderate-budget reference.

4.1 Experimental Setup

We evaluate three open VLMs spanning two decoder lineages—Qwen2.5-VL, Qwen3-VL, and Idefics3-8B-Llama3—on five visual-heavy benchmarks. Four are short-form: DocVQA (Mathew et al., 2021) and InfographicVQA (Mathew et al., 2022) under subtype-level ANLS, ChartQA (Masry et al., 2022) and TextVQA (Singh et al., 2019) under benchmark-native accuracy. The long-context setting is the MMLongBench suite (Wang et al., 2025) at K8, K16, and K32, whose document-QA lane draws on MMLongBench-Doc (Ma et al., 2024); Appendix A.2 lists the subtasks we run and the scored-sample counts behind the combined score. All evaluated methods share each benchmark's model checkpoint, prompt template, decoding parameters, and evaluation harness, ensuring uniform sample coverage and consistent scoring protocols across all comparisons. PACE-VL uses one fixed configuration across the evaluated models and benchmarks.

At 10%, we compare against full-cache decoding (Vanilla) and five compressed baselines: HybridKV, VL-Cache, SparseVLM, CaM (Zhang et al., 2024), and LOOK-M (Wan et al., 2024). CaM adapts the released HELM variant without accumulated attention, using current-step attention and value-only merging. LOOK-M adapts the paper's text-prior selection and pivotal key/value merging to GQA with an exact prompt-cache budget. Appendix A.2 details both ports, including LOOK-M's selection difference from the release. SparseVLM specifies compression by retained token count, so its realized retention ratio can differ from the nominal cache ratio. We report the task metrics above and, for MMLongBench, time to first token (TTFT), decode throughput, and peak allocated memory. Appendix A.2 specifies the scored samples and aggregation rules. Score differences below are in points on a 0–100 scale; the result tables report scores on $[0, 1]$.

4.2 Main Results

At a 10% cache budget, PACE-VL outperforms all five compared compressed baselines in all 21 benchmark–model–scale combinations (Table 1). It matches or exceeds full-cache decoding in 11 settings, while none of the compared compressed baselines reaches the full-cache score. In the remaining 10 settings, its largest deficit is 1.87 points on TextVQA with Qwen3-VL.

The advantage over the strongest competing compressed method ranges from 0.63 to 5.97 points, with a median of 3.06. The largest margins are on DocVQA (5.97 points) and MMLongBench K8 (5.29 points), both with Qwen2.5-VL. Relative to the full cache, PACE-VL matches or exceeds its score in eight of nine MMLongBench settings and three of twelve short-form settings. The remaining long-context setting, K8 on Idefics3, loses 0.34 points. Thus the quality advantage over the compared compressed baselines extends across short-form and long-context tasks, with especially consistent full-cache score retention on MMLongBench.

Table 1: Results at a 10% cache budget. Bold marks the strongest compressed method in each row; † marks rows where PACE-VL matches or exceeds full-cache decoding. Short-form entries use each benchmark's native metric; MMLongBench entries are combined-overall scores at three context scales. CaM and LOOK-M denote the variants and VLM ports in §4.1.

Benchmark	Model	Vanilla	PACE-VL	Hybrid KV	VL-Cache	Sparse VLM	CaM	LOOK-M
Short-form benchmarks								
ChartQA	Qwen2.5-VL	0.8492	**0.8392**	0.7260	0.8040	0.7892	0.6564	0.6624
ChartQA	Qwen3-VL	0.8612	**0.8544**	0.8112	0.8332	0.7944	0.7148	0.6768
ChartQA	Idefics3	0.7320	**0.7392**†	0.6956	0.7228	0.6956	0.5368	0.5532
DocVQA	Qwen2.5-VL	0.8905	**0.8937**†	0.8249	0.8340	0.6684	0.5070	0.6994
DocVQA	Qwen3-VL	0.9571	**0.9523**	0.9350	0.9342	0.7151	0.6410	0.8607
DocVQA	Idefics3	0.7342	**0.7258**	0.6318	0.7163	0.6693	0.4502	0.4529
InfographicVQA	Qwen2.5-VL	0.7373	**0.7346**	0.6857	0.6973	0.6122	0.5661	0.6420
InfographicVQA	Qwen3-VL	0.8000	**0.7915**	0.7767	0.7720	0.6743	0.6638	0.7414
InfographicVQA	Idefics3	0.4816	**0.4805**	0.4238	0.4590	0.4509	0.3674	0.3661
TextVQA	Qwen2.5-VL	0.7359	**0.7301**	0.6344	0.6577	0.6957	0.5103	0.5913
TextVQA	Qwen3-VL	0.8339	**0.8152**	0.7487	0.7697	0.7626	0.5744	0.7225
TextVQA	Idefics3	0.6471	**0.6539**†	0.5980	0.6407	0.6157	0.5261	0.5365
MMLongBench, combined overall								
K8	Qwen2.5-VL	0.5271	**0.5367**†	0.4838	0.4741	0.3860	0.3420	0.3687
K16	Qwen2.5-VL	0.5094	**0.5243**†	0.4767	0.4571	0.3729	0.3263	0.3737
K32	Qwen2.5-VL	0.4596	**0.4754**†	0.4270	0.4019	0.3183	0.2553	0.3122
K8	Qwen3-VL	0.6768	**0.6793**†	0.6473	0.6414	0.5480	0.4885	0.5392
K16	Qwen3-VL	0.6705	**0.6745**†	0.6426	0.6335	0.5414	0.4758	0.5433
K32	Qwen3-VL	0.6530	**0.6554**†	0.6248	0.6201	0.5121	0.4429	0.5111
K8	Idefics3	0.3410	**0.3376**	0.2972	0.3115	0.3313	0.2580	0.2493
K16	Idefics3	0.2771	**0.2820**†	0.2562	0.2668	0.2695	0.2102	0.1993
K32	Idefics3	0.2530	**0.2570**†	0.2283	0.2434	0.2399	0.1631	0.1436

Table 2: Mechanism diagnostics on Qwen3-VL-8B-Instruct: per-setting medians over 20 samples (60 total), first 16 generation decisions, 36 layers, 8 KV heads. **Left:** decode attention mass captured by the leading quantiles of the compressible history. **Right:** for the keep set PACE-VL selects at a 10% budget, the mass falling on positions a selection-only scheme would delete, and the resulting renormalization of the survivors. The long-context settings are the MMLongBench document-QA subset, not the combined suite of Table 1. All mass and share values are percentages; Inflation is a multiplier. The model runs with a full cache throughout.

	Decode attention mass in top q of history				Selection-only residual	
Setting	q=1%	q=5%	q=10%	q=20%	Evicted mass	Inflation
DocVQA	77.29	90.00	93.91	97.02	4.03	1.051×
MMLongBench DocQA K8	75.93	89.11	93.21	96.71	8.08	1.107×
MMLongBench DocQA K32	80.50	90.21	93.86	96.87	7.89	1.103×

A moderate budget. At 50%, PACE-VL also ranks first among the compared compressed methods in all 21 settings, with 19 matching or exceeding the full cache, but its short-form lead is at most 0.64 points and HybridKV and VL-Cache also reach full-cache scores there (Appendix A.4). The 10% and 50% evaluations use different full-cache references and MMLongBench scored sets (Appendix A.2), so they do not form a controlled budget sweep.

4.3 Mechanism Diagnostics and Ablation

Table 2 examines two properties that motivate retaining important entries individually and aggregating the residual history: attention concentration and the mass outside the selected set. These measurements use full-cache decoding on Qwen3-VL-8B-Instruct over 60 samples.

When history entries are ranked by their decode attention, the top 1% captures 76–81% of the mass and the top 10% captures 93–94%. Task-field importance is also concentrated on a small fraction of the history (Appendix A.5).

For the keep set selected by PACE-VL at a 10% budget, the entries that selection alone would discard carry 4.03% of decode attention mass on DocVQA, and 8.08% and 7.89% on MMLongBench DocQA K8 and K32. Removing them renormalizes surviving weights upward by factors of 1.05–1.11. Thus, even when attention is concentrated, the residual history has a measurable aggregate contribution. This motivates reserving slots for residual anchors alongside exact evidence.

Component ablations. On Qwen3-VL-8B, MMLongBench K32, at a 10% cache budget, we compare full PACE-VL against four controls, each removing one element of the design at an identical physical budget: multiplicity-aware decoding, residual aggregation, selected keys at the anchors, and task-field-ranked anchor sites. Full PACE-VL attains the highest equal-weight macro-average over the suite's 20 datasets, 65.25 against 64.09–64.66, and is at or above each control on at least 16 of the 20. The largest single effect is on image counting, where full PACE-VL scores 44.00 and the controls 28.00–34.00. Excluding that dataset, full PACE-VL still leads, 66.37 against 65.98–66.27. Appendix A.8 reports the per-dataset scores.

Query-source ablation. The query-source ablation compares full PACE-VL with Tail32, which uses queries from the final 32 prompt tokens. On Qwen3-VL-8B, MMLongBench K32, at a nominal 10% budget, the reported macro-averages are 65.25 and 64.96, a 0.29-point gap. The difference is concentrated in GovReport (33.28 versus 18.49); full PACE-VL wins on six datasets, ties on seven, and loses on seven. Table 7 lists the per-dataset scores.

4.4 Efficiency

PACE-VL prioritizes quality preservation while maintaining competitive generation latency and peak memory. Table 4 compares seven implementations on identical MMLongBench inputs with 128 generated tokens per sample. Methods retain their own cache-budget policies (Appendix A.3).

Across the evaluated context scales on Qwen3-VL-8B, PACE-VL achieves substantial generation speedups over Vanilla (1.31–$1.49\times$) while maintaining high decode throughput of 18.80–18.81 tokens/s (1.55–$4.32\times$ faster than Vanilla). At K32, compared with competing compressed baselines under identical eager attention backends, PACE-VL achieves lower generation latency than CaM (48.42 s) and LOOK-M (51.55 s), and remains closely competitive with the fastest pruning baseline (within 1% of VL-Cache: 47.66 s vs. 47.29 s) with higher task accuracy (Table 1).

The tradeoff is prefill cost: mean TTFT is 1.98–4.42% higher than Vanilla (4.69 s vs 4.49 s at K8, 40.90 s vs 40.11 s at K32). Mean per-request peak allocated memory is 2.92–11.61% lower than Vanilla, and matches CaM within 0.01% across all three context lengths. Generation latency excludes external input preprocessing, including task-field localization; peak memory includes weights and intermediate tensors as well as the KV cache.

5 Conclusion

PACE-VL offers a training-free approach to VLM cache compression that combines task-field-guided evidence retention, residual aggregation, and multiplicity-aware decoding under an exact prompt-cache budget. Important tokens remain individually accessible, while anchors preserve aggregate residual contributions through value sums and count-aware normalization. At 10% retention, PACE-VL leads the compared compression baselines in all 21 settings across five benchmarks and three VLMs, matching or exceeding full-cache decoding in 11. On Qwen3-VL-8B, PACE-VL also achieves 1.31–$1.49\times$ generation speedups over full-cache decoding with eager attention. These results highlight the value of jointly designing token retention, residual representation, and decode-time reading for aggressive cache compression.

Future work will focus on enabling FlashAttention-based execution of multiplicity-aware decoding and extending PACE-VL to hybrid architectures. In hybrid models, we plan to study layer-wise cache-budget allocation and how compressed KV histories interact with other memory mechanisms.

REPRODUCIBILITY STATEMENT

Section 3 describes the algorithm: the budget rule in §3.2, the task-field score in Eq. (2), the evidence-token/anchor split in Eq. (3), the decode rule in Eq. (4), and the end-to-end procedure in Algorithm 1; the two structural invariants it maintains are stated and proved in Proposition 1. Appendices A.7 and A.3 report the parameter values, query-source rules, cache-budget formulas, and execution settings for the quality and eager efficiency evaluations. Benchmarks, models, metrics, and baseline adaptations are given in §4.1 and Appendix A.2. We plan to release the implementation and reproduction scripts publicly after further refinement and documentation.

AI USE STATEMENT

Generative AI tools were used for manuscript revision and polishing, including language editing, consistency checks, proofreading, LaTeX formatting, and assisting with related work literature discovery. All cited references were independently verified by the authors, and the authors take full responsibility for the contents of the manuscript.

REFERENCES

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations*, 2023.

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. In *Conference on Language Modeling*, 2025.

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 *European Conference on Computer Vision*, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 23958–23967, 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. In *Advances in Neural Information Processing Systems*, 2024.

Xuyang Liu, Xiyan Gui, Yuchao Zhang, and Linfeng Zhang. Mixing importance with diversity: Joint optimization for KV cache compression in large vision-language models. In *International Conference on Learning Representations*, 2026.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, Pan Zhang, Liangming Pan, Yu-Gang Jiang, Jiaqi Wang, Yixin Cao, and Aixin Sun. MMLongBench-Doc: Benchmarking long-context document understanding with visualizations. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.

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, 2022.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. DocVQA: A dataset for VQA on document images. In *IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2200–2209, 2021.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. InfographicVQA. In *IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1697–1706, 2022.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. RazorAttention: Efficient KV cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024. URL https://arxiv.org/abs/2407.15891.

Dezhan Tu, Danylo Vashchilenko, Yuzhe Lu, and Panpan Xu. VL-Cache: Sparsity and modality-aware KV cache compression for vision-language model inference acceleration. In *International Conference on Learning Representations*, 2025.

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

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. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4065–4078, 2024.

Zhaowei Wang, Wenhao Yu, Xiyu Ren, Jipeng Zhang, Yu Zhao, Rohit Saxena, Liang Cheng, Ginny Wong, Simon See, Pasquale Minervini, Yangqiu Song, and Mark Steedman. MMLongBench: Benchmarking long-context vision-language models effectively and thoroughly. *arXiv preprint arXiv:2505.10610*, 2025.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Jian Yuan, Ziwei He, Haoli Bai, Jingwen Leng, and Bo Jiang. WeightedKV: Attention scores weighted key-value cache merging for large language models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2025.

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)*, 2026.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. SparseVLM: Visual token sparsification for efficient vision-language model inference. In *International Conference on Machine Learning*, pp. 74840–74857, 2025.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache merging for memory-efficient LLMs inference. In *International Conference on Machine Learning*, pp. 58840–58850, 2024.

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. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. *arXiv preprint arXiv:2602.16284*, 2026. URL https://arxiv.org/abs/2602.16284.

A Appendix

A.1 Positioning Against Related Methods

Table 3: Design comparison of PACE-VL and representative efficiency methods. "Selection signal" describes the information used to choose entries or head policies; "Fate of unselected" describes their representation after compression. ToMe and RazorAttention use count-based attention correction, while Attention Matching fits logit biases and values to preserve attention mass and outputs.

Method	Stage	Selection signal	Fate of unselected	Budget
StreamingLLM	KV cache	none (position only)	discarded	window + sinks
H2O	KV cache	all past queries	discarded	token count
SnapKV	KV cache	fixed prompt suffix	discarded	token count
PyramidKV	KV cache	attention statistics	discarded	per-layer count
CaM	KV cache	attention statistics	merged into survivors	token count
KVMerger	KV cache	key similarity	kernel-weighted merge	token count
WeightedKV	KV cache	attention statistics	merged into neighbours	token count
RazorAttention	KV cache	retrieval-head probes	count-weighted mean	full / local by head
Attention Matching	KV cache	reference queries	fitted values + biases	target slot count
FastV	decoder layers	received attention	pruned	ratio after layer K
ToMe	ViT blocks	key similarity (no query)	size-weighted merge	merge count
SparseVLM	decoder layers	selected text raters	partly recycled by clustering	adaptive layerwise sparsity
LOOK-M	KV cache (MM)	text prior + attention	merged into retained	token count
VL-Cache	KV cache (MM)	modality-aware score	discarded	layer-adaptive
HybridKV	KV cache (MM)	text-centric, per head	pruned or retrieved	hierarchical
PACE-VL	KV cache	semantic task field	anchored with multiplicity	exact size C

A.2 Evaluation Protocol Details

CaM implementation and reporting. CaM uses the authors' released HELM operator without accumulated attention (`zyxxmu/cam`, commit `fc80e64f460b`), adapted to the three VLMs' grouped-query attention. Query-head scores are averaged within each KV-head group. The cache budget is $\lfloor rT \rfloor$, split equally between initial and recent positions, with the integer remainder assigned to the recent portion. The operator uses current-step attention, merges values but not keys, and retains its fixed merge span of 32 and its short-budget no-op guard; it performs no prefill merging. The short-form runs score all 2500 ChartQA, 5349 DocVQA, 2801 InfographicVQA, and 5000 TextVQA samples. Long-context scores use the same per-budget subtask aggregation and scored counts described below. CaM uses the benchmark evaluator wrappers and the model checkpoints for each family; its implementation is also evaluated in the efficiency study (Table 4).

LOOK-M implementation and reporting. LOOK-M implements text-prior selection and pivotal merging from Wan et al. (2024), with reference to `SUSTechBruce/LOOK-M` (commit `ecf0f51a9c41`). The release targets LLaVA; our port averages causal prompt-attention scores over query heads sharing a KV head and uses each VLM's visual-position masks. With $C_{\mathrm{LOOK}} = \lfloor 0.1T \rfloor$, it retains $\lfloor C_{\mathrm{LOOK}}/2 \rfloor$ high-scoring positions outside the recent window and $C_{\mathrm{LOOK}} - \lfloor C_{\mathrm{LOOK}}/2 \rfloor$ recent positions. Selection follows the paper's maximum-score text prior and exact top-N rule; the released nonnegative-score mask can retain more text than this budget allows. Discarded entries are assigned by key cosine similarity and merged into both retained keys and values using pivotal weights. The prefill output is unchanged; the stored cache is compressed afterward, and ordinary decoding appends exact generated KV entries without multiplicity correction. Every layer starts with C_{LOOK} slots and grows to $C_{\mathrm{LOOK}} + g$ after g cached generated tokens.

The 10% LOOK-M evaluation spans three models and 21 settings, using the same short-form sample counts and MMLongBench scoring conventions as the other methods. Table 4 reports inference costs under the shared efficiency protocol. At 50%, LOOK-M uses $C_{\mathrm{LOOK}} = \lfloor 0.5T \rfloor$ with the same selection and merge rules across the same three models and 21 settings (Table 5). MMLongBench includes `text_doc`; all 50% methods use the same scored subtasks and sample counts.

MMLongBench scope and scored samples. Our MMLongBench runs cover the suite's document-QA group (`longdocurl`, `mmlongdoc`, `slidevqa`), visual RAG (`infoseek`, `viquae`), visual haystacks, needle retrieval, counting and reasoning over both image and text

needles, in-context classification (cars196, food101, inat2021, sun397), summarization (gov-report, multi-lexsum, text_doc), and triviaqa. Combined Overall is the mean over scored samples pooled across these subtasks. At the 10% budget, the text_doc summarization subtask does not enter the combined metric for any method or model, so each scale scores $3551/3651$ samples at K8, $4178/4278$ at K16, and $4414/4514$ at K32, whereas the 50% study, including LOOK-M, scores text_doc and uses all $3651/4278/4514$ samples at K8/K16/K32. Within each budget, all methods use the same scored subtask set.

Baselines use the method-specific budget settings described in §4.1; a token-count setting can yield a different retained fraction from the nominal cache ratio.

A.3 Efficiency Measurements

Table 4: Efficiency of seven implementations on common inputs. Entries are sample means over 3651, 4278, and 4514 inputs on Qwen3-VL-8B at K8, K16, and K32, respectively. TTFT and Gen. are seconds; Dec. is tokens/s; Peak is mean per-request peak allocated memory in GiB. Lower is better except for Dec. All methods share identical eager attention backends. Methods retain their own cache policies; physical budgets are not identical.

	K8				K16				K32			
Method	TTFT	Dec.	Gen.	Peak	TTFT	Dec.	Gen.	Peak	TTFT	Dec.	Gen.	Peak
Qwen3-VL-8B												
Vanilla	4.49	12.15	15.03	18.055	12.81	7.60	29.82	20.006	40.11	4.36	70.19	24.301
PACE-VL	4.69	18.80	11.45	17.528	13.19	18.81	19.95	18.832	40.90	18.80	47.66	21.481
HybridKV	6.90	22.22	12.63	18.070	17.37	22.02	23.14	20.020	49.13	18.28	56.09	24.327
VL-Cache	4.61	22.17	10.35	18.064	12.96	21.98	18.74	20.020	40.33	18.27	47.29	24.327
SparseVLM	4.51	20.48	11.08	18.059	12.86	20.02	20.26	20.015	40.19	19.64	49.40	24.317
CaM	4.52	17.34	11.86	17.527	12.85	17.16	20.28	18.831	40.19	15.56	48.42	21.479
LOOK-M	5.05	19.94	11.44	17.582	14.15	19.14	20.80	18.838	43.87	16.57	51.55	21.482

Execution and scope. Methods are measured sequentially per sample on the same GPU, rotating method order and clearing generation caches and attention state. Inputs and physical GPU assignments are shared, and every method uses BF16, identical chunked prefill (text/vision query chunks $128/256$), warmup, identical eager attention decoding, and greedy generation forced to 128 tokens. The 21 subtasks include text_doc; each sample has equal weight, so larger datasets contribute more to the means. The quality scores in Table 1 exclude text_doc.

Method configurations. The PACE-VL implementation evaluated in Table 4 scores history using task-field queries (§3.2). It uses max pooling ($k = 5$), threshold $\tau = 3.5 \times 10^{-5}$, and dynamic evidence clamp fractions $(0.55, 0.75)$. For prompt length T, the total prompt-cache budget follows §3.2 with $r = 0.1$, $R = 32$, and $C_{\min} = 64$:

$$C = \min(T, \max(64, R + 1, \lfloor 0.1T \rfloor)) .$$

Under active compression, the protected R recent entries are included within C; the remaining $C-R$ slots are split between evidence tokens and anchors by Eq. (3). The run uses eager multiplicity-aware decoding and appends generated KV entries. LOOK-M initially retains $\lfloor 0.1T \rfloor$ entries; both methods append generated KV entries. CaM uses its released HELM variant without accumulated attention and maintains a fixed start/recent window through decode-time merging and eviction. VL-Cache and HybridKV use ratio 0.1 with their layer/head policies; SparseVLM uses the 192-token setting and pruning stages $[2, 6, 15]$.

Measurement and aggregation. GPU synchronization brackets generation and first-token emission. TTFT includes vision/text prefill, compression, cache conversion, and the first token; decode throughput is $127/(t_{\text{end}} - t_{\text{first}})$ per sample, then averaged. Gen. spans generate() entry to synchronized completion, excluding input preprocessing, loading, warmup, and allocator cleanup. Peak counters reset per request; allocated memory is converted from MiB to GiB and includes resident weights and device inputs. Entries are arithmetic means from one measurement per method and sample. Generation speedups divide baseline mean latency by PACE mean latency; decode ratios divide mean throughputs.

A.4 Results at a Moderate Budget

Table 5 gives the full results of the 50%-budget evaluation summarized in §4.2.

At 50%, PACE-VL ranks first among the compared compressed methods in all twenty-one combinations and matches or exceeds full-cache decoding in nineteen. The evaluation comprises twelve short-form combinations and nine MMLongBench blocks.

On short-form tasks, PACE-VL, HybridKV, and VL-Cache all finish within 0.62 points of the full cache in every combination. PACE-VL leads the best competing method by 0.04–0.64 points, compared with a maximum lead of 5.97 points at the 10% budget. HybridKV or VL-Cache match or exceed the full cache in eight of the twelve combinations, by at most 0.24 points. SparseVLM trails the full cache by a median of 6.8 points and by as much as 28.5 points on these tasks.

On MMLongBench, PACE-VL exceeds the full cache in eight of nine blocks. LOOK-M also exceeds it on Qwen2.5-VL at K32, while the other baselines remain below it in all blocks. SparseVLM's median gap to the full cache is 12.5 points.

Table 5: Results at a 50% cache budget. Bold marks the strongest compressed method in each row; † marks rows where PACE-VL matches or exceeds full-cache decoding. Short-form entries use each benchmark's native metric. MMLongBench entries are sample-weighted combined-overall scores; all methods, including LOOK-M, score $3651/4278/4514$ samples at K8/K16/K32, including `text_doc`. CaM and LOOK-M denote the variants and VLM ports in §4.1 and Appendix A.2.

Benchmark	Model	Vanilla	PACE-VL	HybridKV	VL-Cache	Sparse VLM	CaM	LOOK-M
Short-form benchmarks								
ChartQA	Qwen2.5-VL	0.8468	**0.8496**†	0.8460	0.8492	0.7860	0.8172	0.8484
ChartQA	Qwen3-VL	0.8624	**0.8644**†	0.8640	0.8632	0.7960	0.8172	0.8584
ChartQA	Idefics3	0.7104	**0.7116**†	0.7112	0.7104	0.6752	0.6888	0.7108
DocVQA	Qwen2.5-VL	0.8971	**0.9003**†	0.8912	0.8939	0.6262	0.7883	0.8892
DocVQA	Qwen3-VL	0.9604	**0.9603**	0.9596	0.9598	0.6755	0.8935	0.9595
DocVQA	Idefics3	0.7464	**0.7493**†	0.7417	0.7470	0.6760	0.6351	0.7347
InfographicVQA	Qwen2.5-VL	0.7807	**0.7852**†	0.7773	0.7792	0.6523	0.6609	0.7682
InfographicVQA	Qwen3-VL	0.8294	**0.8306**†	0.8293	0.8298	0.7104	0.7356	0.8219
InfographicVQA	Idefics3	0.4738	**0.4759**†	0.4717	0.4738	0.4421	0.4173	0.4601
TextVQA	Qwen2.5-VL	0.7363	**0.7397**†	0.7301	0.7363	0.6978	0.6950	0.7297
TextVQA	Qwen3-VL	0.8348	**0.8356**†	0.8320	0.8333	0.7617	0.7956	0.8349
TextVQA	Idefics3	0.6419	**0.6473**†	0.6412	0.6427	0.6079	0.6115	0.6397
MMLongBench, combined overall								
K8	Qwen2.5-VL	0.4874	**0.4914**†	0.4821	0.4633	0.3517	0.3742	0.4804
K16	Qwen2.5-VL	0.4675	**0.4700**†	0.4641	0.4497	0.3303	0.3743	0.4581
K32	Qwen2.5-VL	0.4014	**0.4165**†	0.3978	0.3836	0.3176	0.3315	0.4056
K8	Qwen3-VL	0.6677	**0.6683**†	0.6634	0.6510	0.5427	0.5127	0.6553
K16	Qwen3-VL	0.6464	**0.6480**†	0.6444	0.6355	0.5209	0.5054	0.6301
K32	Qwen3-VL	0.6247	**0.6249**†	0.6206	0.6154	0.4852	0.4762	0.5981
K8	Idefics3	0.3424	**0.3394**	0.3293	0.3374	0.3350	0.2631	0.2821
K16	Idefics3	0.2792	**0.2811**†	0.2737	0.2387	0.2291	0.2128	0.2279
K32	Idefics3	0.2549	**0.2672**†	0.2523	0.1774	0.1609	0.1604	0.1668

A.5 Task-Field Importance Polarization

Table 2 ranks the compressible history by the attention it actually receives at decode time. Table 6 repeats the measurement with the history ranked instead by the task-field importance of Eq. (2), which is the signal PACE-VL uses to build the cache. The two profiles agree to within about three points at every quantile, so both concentrate on comparably small subsets of the history.

Both diagnostic tables use 20 samples per setting, drawn with seed 42 from prompts whose 10% cache budget exceeds the implementation's 64-token minimum.

Table 6: Share of task-field importance mass captured by the leading quantiles of the compressible history; per-setting medians over the same 20 samples as Table 2. All values are percentages.

Setting	q=1%	q=5%	q=10%	q=20%
DocVQA	78.05	90.00	93.86	96.72
MMLongBench DocQA K8	72.82	86.17	91.00	95.14
MMLongBench DocQA K32	79.14	88.03	91.84	95.32

A.6 Deferred Proofs

Proof of Proposition 2. By Proposition 1 the sets $\mathcal{A}_a$ partition $\{1, \dots, T\}$, with singletons for evidence tokens and recent entries. Grouping the sums in Eq. (1) by part and substituting $k_i = k_a$ gives numerator $\sum_a \exp(q^\top k_a/\sqrt{D}) \sum_{i \in \mathcal{A}_a} v_i = \sum_a \exp(q^\top k_a/\sqrt{D})\, S_a$ and denominator $\sum_a \exp(q^\top k_a/\sqrt{D})\, |\mathcal{A}_a| = \sum_a \exp(q^\top k_a/\sqrt{D})\, n_a$, which is Eq. (4). □

Proof of Proposition 3. Substituting $\exp(q^\top k_i/\sqrt{D} + \delta_i) = \exp(q^\top k_{a(i)}/\sqrt{D})$ into Eq. (1) and grouping by anchor reproduces Eq. (4) exactly as in the proof of Proposition 2. The stated bound on $|\delta_i|$ is Cauchy–Schwarz applied to $q^\top (k_{a(i)} - k_i)$. Since $\|q\|/\sqrt{D}$ does not depend on the assignment, minimizing the bound for token i means minimizing $\|k_{a(i)} - k_i\|$, which is the nearest-anchor rule in key space. □

Proof of Proposition 1. The cache holds $N_{\text{evid}} + N_{\text{anchor}} = C - R$ history entries by Eq. (3) and R recent entries, and the budget rule in §3.2 guarantees $C > R$, so its length is exactly C. For the coverage identity, Eq. (3) guarantees at least one anchor, so the assignment step is well defined and sends each of the $T - R$ history positions to exactly one destination: itself if it is an evidence token, otherwise its nearest anchor. Evidence-token and anchor multiplicities therefore sum to $T - R$, and the R recent entries contribute one each, giving $\sum_a n_a = T$. □

A.7 Implementation Details

PACE-VL is specified by the cache ratio r, the protected recent window R, the minimum cache size $C_{\min}$, the pooling kernel k, the task-field importance threshold τ, and the clamp fractions $(\alpha_{\min}, \alpha_{\max})$. The task-field reference configuration uses $r = 0.1$, $R = 32$, $C_{\min} = 64$, max pooling with $k = 5$, importance threshold $\tau = 3.5 \times 10^{-5}$, and evidence clamp fractions $(\alpha_{\min}, \alpha_{\max}) = (0.55, 0.75)$. The nominal ratio is $r = 0.5$ in the moderate-budget evaluation.

The quality, eager efficiency, and full-method ablation evaluations use the complete textual task field as the query source: question text is aligned to its prompt-token positions, and summarization tasks use their explicit summary instruction (§3.2). The whole-prompt budget follows §3.2, including the protected recent window; the evidence/anchor split follows Eq. (3). Component controls share this budget. The Tail32 query-source ablation replaces the task-field queries with the final 32 prompt tokens while retaining the same budget convention.

A.8 Component Ablations

Table 7 reports the experiment summarized in §4.3: full PACE-VL and four controls on Qwen3-VL-8B over MMLongBench K32 at a 10% cache budget. All five arms occupy the same physical budget, so the controls change what fills the slots rather than how many there are. *Without multiplicity* retains the same keys, cluster assignments, and mean values $\bar{v}_a = S_a/n_a$ as the full method, but removes the additive $\log n_a$ bias from the attention logits (§3.4). It therefore applies ordinary softmax attention to the cluster means, isolating the effect of multiplicity-based weighting. *All-evidence* fills every non-recent slot with an original key–value pair of multiplicity one and evicts the residual history instead of aggregating it. *Centroid keys* stores a coordinate-wise mean key at each anchor in place

of a selected token's key. *Random anchors* draws the anchor sites uniformly at random instead of taking the highest-scoring positions left after the evidence slots are filled, under a single seed.

Scores are equal-weight macro-averages over the suite's 20 datasets, $S_{\text{macro}} = \frac{1}{20}\sum_{d=1}^{20} s_d$ with s_d on a 0–100 scale. Table 1 reports a sample-weighted overall score (65.54 for Qwen3-VL at K32), whereas the full-method reference here is a dataset-level macro-average (65.25). These metrics assign different weights to datasets.

Tail32 query-source comparison. Tail32 uses queries from the final 32 prompt tokens. Full PACE-VL scores 0.29 points higher overall, but 0.48 points lower across the 19 datasets excluding GovReport; the difference is task-dependent.

Table 7: Component and query-source comparisons on Qwen3-VL-8B, MMLongBench K32, at a nominal 10% budget. The four component controls match the full method's physical cache size. Tail32 uses queries from the final 32 prompt tokens. Scores use a 0–100 scale and equal-weight macro-averaging; bold marks the highest macro-average.

		Component control				
Dataset	Full PACE-VL	w/o multiplicity	All-evidence	Centroid keys	Random anchors	Tail32 queries
infoseek	45.00	45.00	45.00	44.00	46.00	46.00
viquae	56.00	57.00	55.00	56.00	56.00	55.00
Visual haystacks, single	47.00	47.00	47.00	47.00	47.00	47.00
Visual haystacks, multi	51.00	51.00	51.00	51.00	51.00	50.00
NIAH image retrieval	86.00	86.00	86.00	86.00	86.00	86.00
NIAH image counting	44.00	30.00	28.00	32.00	34.00	44.00
NIAH image reasoning	68.00	68.00	68.00	68.00	68.00	70.00
NIAH text retrieval	100.00	100.00	100.00	100.00	100.00	100.00
NIAH text counting	36.00	34.00	34.00	36.00	36.00	38.00
NIAH text reasoning	62.00	61.00	62.00	63.00	62.00	66.00
cars196	100.00	100.00	100.00	100.00	100.00	100.00
food101	94.00	93.00	93.00	94.00	94.00	94.00
inat2021	83.00	83.00	84.00	83.00	82.00	85.00
sun397	98.00	97.00	97.00	97.00	97.00	97.00
gov-report	33.28	32.55	32.45	33.03	33.25	18.49
multi-lexsum	21.57	21.06	21.20	22.20	23.04	21.35
triviaqa	93.00	91.00	91.00	91.00	93.00	93.00
longdocurl	61.60	61.34	61.38	61.45	60.47	62.79
mmlongdoc	52.98	53.43	52.86	53.06	51.85	52.91
slidevqa	72.56	72.85	72.82	72.80	72.49	72.75
Macro-average	**65.25**	64.21	64.09	64.53	64.66	64.96
Δ vs. full	—	−1.04	−1.16	−0.72	−0.59	−0.29

Full PACE-VL matches or exceeds the four component controls on 17, 18, 16, and 18 of 20 datasets, respectively, in table order. Six datasets tie across these five arms. Image counting contributes gaps of 10–16 points, and text counting up to 2. Excluding image counting, full PACE-VL still leads these controls: 66.37 versus 66.01, 65.98, 66.24, and 66.27. Random anchors use a single seed.

A.9 Limitations

Question position is a scope condition. Attention is causal, so a probe at position f scores only history at positions $i \leq f$ (Eq. 2): evidence appearing *after* the entire task field receives no task-field signal. Our layouts place the visual evidence first, which is what the method is built for; a question-first prompt would leave the later evidence unranked and need a different probe construction, such as re-running the task-field queries at the end. PACE-VL also assumes a runtime adapter can locate the task field at all—a prompt without one, or whose task is spread across a long dialogue, is out of scope—and we have not measured sensitivity to misidentified fields.

Within-cluster key variation. Proposition 3 bounds the decode perturbation by each cluster's key-space radius. Larger within-cluster key differences loosen this bound. Individually retained evidence tokens keep their original keys and values, avoiding key replacement.

Prefill overhead. Anchor assignment is quadratic in prompt length at a fixed budget ratio (§3.5). In the efficiency study on Qwen3-VL-8B, mean TTFT increases by 1.98–4.42% relative to Vanilla despite faster fixed-length generation. The tradeoff depends on model, input length, and output length; shorter outputs provide less opportunity to amortize cache construction.