PAGEPRUNE: BRIDGING THE PRUNING–EXECUTION GAP IN PAGED-ATTENTION KV CACHE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context large language model (LLM) inference is increasingly constrained by the growing Key-Value (KV) cache, which consumes substantial GPU memory and incurs repeated memory traffic during decoding. Existing KV-cache pruning methods reduce the retained cache size, but often overlook how the surviving KV entries are organized and accessed by attention kernels, limiting the resulting end-to-end (E2E) gains. We present PagePrune, an execution-aware, page-granular KV-cache pruning system for vLLM that jointly optimizes KV selection and execution layout. PagePrune periodically ranks KV pages using exponentially smoothed attention importance and retains high-utility pages under a configurable budget, while protecting critical prefix and recent context. It further aligns pruning decisions with attention-kernel tiles and performs compact-once repacking to expose the retained cache as a contiguous working set. We integrate PagePrune into vLLM and evaluate it across long-context language, vision question answering, and video-language workloads. Empirically, PagePrune substantially reduces KV-cache memory footprint while achieving up to 7.91$\times$ E2E speedup over vanilla vLLM with comparable task quality. Experiments and analysis demonstrate that effective KV-cache pruning requires not only selecting what to retain, but also organizing how the retained cache is executed.

1 INTRODUCTION

Improving the throughput of large language model (LLM) inference is a persistent objective of modern serving systems (Liu et al., 2025; 2024b; Kwon et al., 2023), where memory efficiency has become as critical as compute efficiency (Liu et al., 2024c; Lin et al., 2025). During autoregressive (AR) decoding, the key-value (KV) cache grows with both context length and the number of concurrent requests, consuming substantial GPU memory and directly constraining the batch size that can be sustained during serving (Yu et al., 2022; Kwon et al., 2023). PagedAttention (Kwon et al., 2023) addresses this bottleneck by partitioning the KV cache into fixed-size pages that are dynamically allocated through per-sequence block tables, significantly reducing the fragmentation and over-reservation associated with contiguous KV allocation. This design largely decouples KV-cache growth from contiguous physical-memory allocation, making dynamic KV management more memory-efficient under heterogeneous serving workloads (Chitty-Venkata et al., 2026; Tang et al., 2024).

Recent work has extended KV-cache optimization beyond allocation toward reducing, reusing, and more efficiently executing the decode-time working set. Specifically, prefix- and cross-request reuse methods improve cache utilization (Zheng et al., 2024; Liu et al., 2025; Gim et al., 2024), while KV-pruning methods retain only a subset of historical states. Token-level approaches such as H2O (Zhang et al., 2023), StreamingLLM (Xiao et al., 2024b), SnapKV (Li et al., 2024), and PyramidKV (Cai et al., 2024) exploit attention concentration, recency, or heavy-hitter patterns, whereas page-oriented methods such as Quest (Tang et al., 2024) and PagedEviction (Chitty-Venkata et al., 2026) operate at a coarser KV-cache granularity. In parallel, systems such as vAttention (Prabhu et al., 2025) and FlashInfer (Ye et al., 2025) optimize the underlying memory and execution paths. However, reducing the logical KV working set does not by itself guarantee proportional decode-time speedup. When retained pages are scattered across the sequence, page-table lookups and weaker memory locality further erode the bandwidth savings. For example, PagedAttention's non-contiguous layout introduces address translation and locality overheads (Prabhu et al., 2025) and kernels can waste many

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.

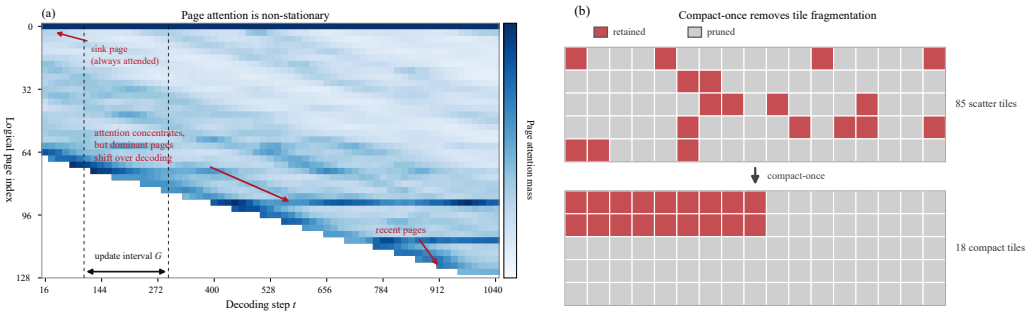

Figure 1: Motivations for PagePrune. Panel (a): Page importance is non-stationary. We record the per-page attention mass of Qwen3-4B at every decode step on four long-generation LongBench tasks; mass concentrates on the sink page, a small set of salient pages, and a sliding recent band, but the salient pages migrate as decoding proceeds, so a keep-list frozen early discards 56–71% of the mass that later steps attend to. Panel (b): The pruning–execution gap at tile granularity. Paged attention reads fixed 8-page tiles, so scattered retained pages force 85 tile reads (18 retained +67 pruned) while compact-once repacking reads only 18 tiles (4.7× fewer).

of KV reads on pages that contribute little to the output (Chitty-Venkata et al., 2026). We refer to this mismatch between page-level sparsity and kernel execution efficiency as the *pruning–execution gap*.

To examine how these issues arise in practice, Figure 1 reports two observations from representative long-context and multimodal workloads. First, page importance is non-stationary. On Qwen3-4B (Yang et al., 2025) over long-generation LongBench (Bai et al., 2023) workloads, attention mass concentrates on a small subset of pages, yet the salient set gradually shifts as decoding proceeds. A keep-list fixed early can therefore become increasingly stale, motivating periodic re-scoring rather than one-shot pruning. Second, page-level pruning alone does not necessarily yield an execution-efficient active cache. On Qwen3-VL-4B-Instruct (Bai et al., 2025a) over ActivityNetQA (Yu et al., 2019), the retained pages span 85 KV tiles in their original logical layout, where a KV tile denotes a fixed group of pages that the attention backend reads as one unit, whereas compacting the same retained set reduces the active span to 18 tiles, corresponding to 4.7× fewer tile accesses. Together, these observations motivate a pruning policy that adapts to evolving page importance while reorganizing the retained pages into a more compact execution layout.

In this paper, we propose PagePrune, a runtime KV-cache management framework that treats the KV page as the basic unit for both redundancy reduction and cache execution. The key idea is to identify and retain only the pages that remain useful for decoding, while continuously reorganizing the active page set as attention patterns evolve. Specifically, PagePrune periodically re-ranks pages using attention mass accumulated directly within the decode attention kernel (Ghadia et al., 2025), allowing the retained set to adapt throughout generation without an additional query–key scoring pass. It further compacts the retained block-table entries into a shorter active layout, reducing the number of KV tiles, traversed by subsequent decoding steps while pruning the underlying physical KV blocks. Together, these mechanisms turn page-level KV redundancy into a dynamically maintained and execution-efficient active cache. PagePrune achieves up to 3.86× speedup over vLLM on long-context language workloads, 5.22× on multimodal question answering, and 7.91× on video-language tasks, while substantially reducing KV-cache memory and maintaining comparable task quality. These results across diverse long-context workloads demonstrate that KV pruning alone does not guarantee E2E acceleration: aligning the retained cache with the execution layout is critical for translating cache reduction into practical serving gains. Our contributions are as follows:

- Attention-based Keep/Drop page management. We dynamically classify KV pages into KEEP and DROP using accumulated attention scores, capturing the evolving page utility during decoding.
- Tile-aligned compact-once execution. We align retained pages with the attention kernel's KV-tile granularity and compact them into a contiguous layout, translating logical KV reduction into more efficient attention execution.
- Native vLLM implementation and broad validation. We implement PagePrune natively in vLLM and validate it across diverse model families, scales, modalities, benchmarks, and serving settings, demonstrating substantial memory and E2E efficiency gains.

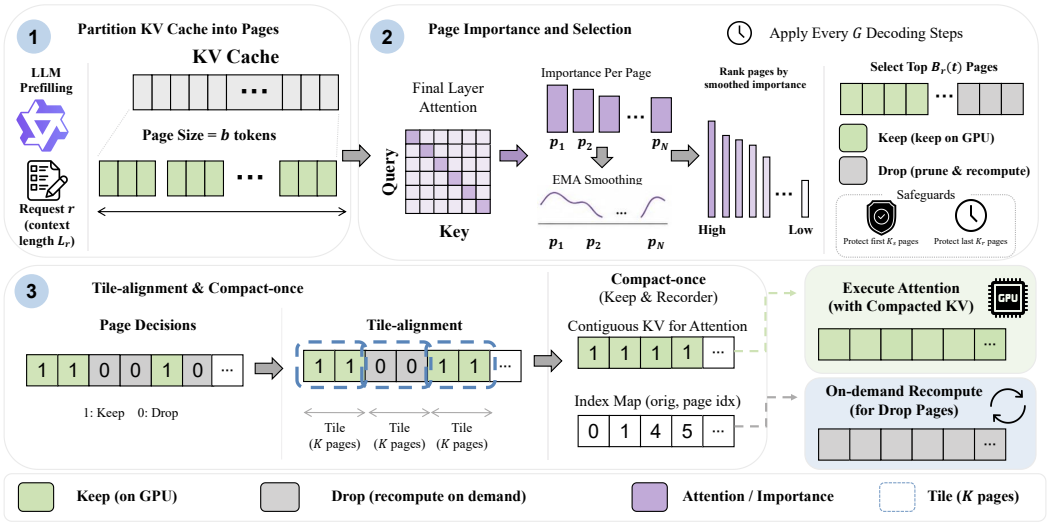

Figure 2: Overview of PagePrune. Every G decoding steps, PagePrune estimates page importance from decode-time attention, selects pages under a target retention budget with sink and recency protection, and aligns the resulting decisions with the KV-tile granularity of the attention backend. Pages outside the tile-aligned retained set are removed from the active KV cache, after which the block table is compacted over the surviving pages for subsequent decoding.

2 Background

AR decoding and KV caching. At decoding step t, a transformer layer computes the query, key, and value vectors $q^{(t)}, k^{(t)}, v^{(t)}$ for the current token. The current query attends to all key and value vectors available up to the current position: $K_{<t} = [k^{(1)}, \ldots, k^{(t)}], V_{<t} = [v^{(1)}, \ldots, v^{(t)}]$. The attention probabilities and output are computed as $\boldsymbol{\alpha}^{(t)} = \mathrm{softmax}\left(\frac{q^{(t)}(K_{<t})^\top}{\sqrt{d}}\right)$, $o^{(t)} = \boldsymbol{\alpha}^{(t)} V_{<t}$, where $\alpha_i^{(t)}$ denotes the attention probability assigned to token i at decoding step t. After the attention computation, $(k^{(t)}, v^{(t)})$ is retained in the KV cache and reused in subsequent decoding steps.

Paged KV cache. PagedAttention (Kwon et al., 2023) organizes the KV cache into fixed-size logical pages rather than requiring each request to occupy a contiguous physical region. For request r at decoding step t, let $\mathcal{P}_r(t) = \{p_1, p_2, \ldots, p_{N_r(t)}\}$ denote the logical page sequence of its retained KV cache, where each page p_i contains up to b consecutive tokens and $N_r(t) = \lceil L_r(t)/b \rceil$ for context length $L_r(t)$. The KV cache is maintained per layer and per attention head. A block table $\phi_r^{(t)} : \{1, \ldots, N_r(t)\} \to \mathcal{B}$ maps each logical page p_i to a physical KV block $\phi_r^{(t)}(i)$ in the global GPU memory pool $\mathcal{B}$. Logical adjacency therefore does not imply physical adjacency: consecutive pages of a request may reside in arbitrary physical blocks. PagePrune operates on this logical-page abstraction while modifying which corresponding physical KV blocks remain resident and how the retained pages are presented to the attention backend.

3 PagePrune

PagePrune is a runtime KV-cache pruning policy built on the paged KV-cache abstraction of PagedAttention (Kwon et al., 2023). For request r at decoding step t, let $\mathcal{P}_r(t) = \{p_1, \ldots, p_{N_r(t)}\}$ denote its logical page sequence as defined in Section 2, and let $\mathcal{C}_r(t) \subseteq \mathcal{P}_r(t)$ denote the pages whose KV blocks are currently retained in the active cache. As illustrated in Figure 2, every G decoding steps, PagePrune first restores the pages pruned in the previous interval, re-estimates their importance from the current attention computation, selects a subset to retain, aligns the selection with the execution granularity of the attention backend, and removes the remaining KV blocks. The corresponding block table is then rebuilt over the retained pages and used until the next update. A tile is a fixed group of consecutive pages that the attention backend accesses as a single unit, and PagePrune aligns its retention decisions to this tile granularity.

Page Importance Estimation and Selection. To determine which KV pages can be removed while preserving the states most relevant to decoding, PagePrune begins by estimating page importance. Here, attention provides a natural signal, as it directly reflects how strongly the current decoding query relies on the cached tokens within each page. Since these attention probabilities are already computed during decode-time attention, PagePrune can derive page importance without introducing an additional query–key scoring pass.

Let $\alpha_{h,i}^{(t)}$ denote the attention probability assigned by head h of the final transformer layer to cached token i at decoding step t. We first average the probabilities across the H attention heads, $\alpha_i^{(t)} = \frac{1}{H}\sum_{h=1}^{H}\alpha_{h,i}^{(t)}$, and define the attention mass of page p as $a_p^{(t)} = \sum_{i\in p}\alpha_i^{(t)}$. PagePrune accumulates these page-level statistics during the existing attention computation and therefore does not require an additional query–key scoring pass. However, attention patterns can fluctuate across decoding, making a single-step observation potentially unreliable for page retention. PagePrune therefore maintains an exponentially smoothed importance score to capture more stable attention trends over time:

$$\bar{A}_p^{(t)} = \mu a_p^{(t)} + (1-\mu)\bar{A}_p^{(t-1)}, \tag{1}$$

where $\mu \in (0,1)$ controls the contribution of the current observation. PagePrune updates the smoothed scores once every G decoding steps, initializing pages allocated since the previous update with the current attention mass. Eviction releases the KV blocks of the pages but retains their smoothed importance scores, allowing their historical attention information to be carried forward and updated when the pages are restored. At each update, it sets a target budget of $B_r(t) = \lceil \tau N_r(t) \rceil$, where $\tau \in (0,1)$ is the target retention ratio, and ranks pages according to $\bar{A}_p^{(t)}$. Additionally, we protect the first K_s logical pages and the most recent K_r logical pages to avoid removing positions that are typically important during decoding, as suggested by Zhang et al. (2023). The provisional retained set is

$$R_r(t) = \mathrm{Top}_{B_r(t)}\big(\bar{A}\big) \cup \{p_1, \ldots, p_{K_s}\} \cup \{p_{N_r(t)-K_r+1}, \ldots, p_{N_r(t)}\}. \tag{2}$$

Here, $\mathrm{Top}_{B_r(t)}(\bar{A})$ denotes the highest-scoring pages among the currently active pages, including pages restored at the beginning of the current pruning interval. Because the protected pages are added to the ranking-selected set, the resulting number of retained pages may exceed $B_r(t)$. Pages outside $R_r(t)$ are treated as pruning candidates, while the final decision is made after tile alignment. To improve GPU memory utilization, PagePrune prunes these pages from the active KV cache, with the final pruning decision determined after tile alignment. The corresponding KV blocks are released back to the memory pool once the drop page set is finalized.

Tile-Aligned Eviction and Block-Table Compaction. Although PagePrune selects pages according to their importance, the attention backend processes the KV cache at a coarser tile granularity. Here, a tile refers to the execution granularity of the attention backend; PagePrune models it as K consecutive logical pages, corresponding to Kb tokens. Directly applying page-level eviction may therefore leave partially retained tiles and introduce irregular memory accesses. PagePrune instead expands the retention decisions to the backend's KV-tile boundaries before modifying the cache. Let $T(p)$ denote the tile containing page p. Given the page-level decision $m_p = \mathbb{I}[p \in R_r(t)]$, PagePrune derives the tile-aligned decision as $\tilde{m}_p = \bigvee_{j\in T(p)} m_j$, so that selecting any page in a tile preserves the entire tile. The resulting decisions define the final retained set $\tilde{R}_r(t) = \{p \in \mathcal{C}_r(t) \mid \tilde{m}_p = 1\}$. PagePrune then removes only the pages in $\mathcal{C}_r(t) \setminus \tilde{R}_r(t)$ from the active KV cache and releases their physical blocks. Because a single selected page may preserve additional pages within the same tile, the effective retention ratio after alignment can exceed the target τ.

After eviction, PagePrune rebuilds the request's block table using the physical block identifiers associated with pages in $\tilde{R}_r(t)$, preserving their original logical order. It also retains the corresponding logical page indices so that each compacted block-table entry can be mapped back to its position in the original sequence. This operation compacts the block-table representation; it does not require the surviving physical KV blocks themselves to occupy contiguous GPU addresses. Subsequent decoding traverses only the retained block-table entries, without evaluating a page-level keep mask at every decoding step. The complete score–select–align–compact procedure is repeated every G steps, amortizing its update overhead across the intervening decoding steps. PagePrune recomputes the KV states of pages pruned in the previous interval from their stored token inputs and the corresponding context, restoring them to the active cache before the next pruning decision. The complete procedure is summarized in Algorithm 1 in Appendix B.

4 EXPERIMENTS

Benchmarks. We evaluate PagePrune on long-context language and multimodal tasks that jointly stress KV-cache capacity and attention-execution efficiency. For language, we use RULER (Hsieh et al., 2024) (synthetic retrieval and reasoning), LongBench (Bai et al., 2023) (diverse realistic tasks), InfiniteBench (Zhang et al., 2024) (substantially longer contexts), and LongBench-v2 (Bai et al., 2024) (challenging multi-hop understanding); single-fact retention is further probed with a needle-in-a-haystack (NIAH) test whose full retrieval maps are in Appendix G. For multimodality, we use video question answering (VQA) task: NExTQA (Xiao et al., 2021), ActivityNet-QA (ActQA) (Yu et al., 2019), and EgoSchema (Mangalam et al., 2023), and video summarization (VS) by ActivityNet Captions (ActCap) (Krishna et al., 2017) and DREAM1K (Wang et al., 2024). These benchmarks evaluate whether PagePrune preserves accuracy as context length and task complexity increase.

Metrics. We report task quality, peak GPU memory, generation throughput, and end-to-end (E2E) latency. For language and VQA, we consider accuracy (*Acc*), while VS uses ROUGE-L. Peak GPU memory (*Mem*) measures the maximum GPU memory used during inference. Generation throughput (*Thp*) is the aggregate decode rate, computed as the total number of output tokens generated across all evaluated requests divided by the total wall-clock time spent in the decode phase. E2E latency (*Latency*) measures the complete request latency, including computation and KV-cache management overhead. To make speedups comparable across methods, we report the relative speedup over vanilla vLLM: $\mathrm{RS} = \frac{\mathrm{Latency}_{\mathrm{vLLM}}}{\mathrm{Latency}_{\mathrm{method}}}$, and larger RS shows more significant acceleration.

Baselines. We compare PagePrune against representative baselines spanning the KV-cache serving stack. We use vanilla vLLM (Kwon et al., 2023) as the reference serving system, and include LMCache (Liu et al., 2025), which reduces long-context memory pressure via KV-cache compression and memory-tier management. For KV-cache selection and eviction, we compare with Quest (Tang et al., 2024), which identifies important KV pages for attention but does not physically remove unselected entries from GPU memory, and PagedEviction (Chitty-Venkata et al., 2026), which physically evicts unselected pages while retaining the standard paged-attention execution path. We additionally include Random Attention (Wang et al., 2026a) as an attention-based KV selection baseline, with its results reported in the Appendix D. These baselines separate logical KV selection, physical cache reduction, and execution-efficient cache organization.

Implementation Details. PagePrune is implemented on vLLM v0.23.1rc0's[1] native paged KV-cache. Each page stores 16 consecutive tokens, and retained pages are organized into execution-aligned groups of 8 pages to match the granularity of the underlying attention kernel and avoid fragmented accesses. We update page importance with an EMA using $\mu = 0.5$ and set the default retention ratio to $\tau = 0.3$. The first page is always retained as an attention sink, and the two most recent pages are protected for recency. Unless otherwise specified, pruning decisions are refreshed every $G = 64$ decoding steps. We evaluate LLMs Qwen3-4B and Qwen3-8B (Yang et al., 2025) and VLLMs Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct (Bai et al., 2025a), with Qwen2.5-32B-Instruct and LLaVA-NeXT-Video-7B (Bai et al., 2025b; Liu et al., 2024a) included in Appendix C. Unless otherwise specified, we use a serving batch size of 32 for all inference experiments. All experiments run on NVIDIA Pro 6000 GPUs with 95 GB memory per GPU. Latency and throughput are averaged over three runs. For fair comparison, all methods use the same serving and decoding configurations, and all shared experimental hyperparameters are kept identical across PagePrune and the baselines; method-specific hyperparameters are set according to their respective implementations. We report the run-to-run stability of accuracy across these repeated runs in Appendix E.

4.1 MAIN RESULTS

Long-context language. Table 1 evaluates PagePrune across four long-context language benchmarks with Qwen3-4B and Qwen3-8B. PagePrune consistently improves E2E serving efficiency across diverse workloads, although the magnitude of the gain varies with the workload and model size. It achieves up to $2.38\times$ speedup on RULER, $3.59\times$ on LongBench, $2.18\times$ on InfiniteBench, and $3.03\times$ on LongBench-v2 for Qwen3-4B, with comparable accuracy on most settings. For Qwen3-8B, PagePrune achieves $3.86\times$, $2.95\times$, $2.71\times$, and $2.94\times$ speedups on the four benchmarks, respectively. The comparison with Quest and PagedEviction further highlights that reducing the logical KV

[1] Released on 2026-06-15.

Table 1: Long-context Language inference performance on RULER, LongBench (LB), InfiniteBench (InfB), and LongBench-v2 (LB-v2) using Qwen3-4B and Qwen3-8B.

	Method	Qwen3-4B					Qwen3-8B				
		Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑	Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑
RULER	vLLM	37.51	90.97	31.11	9.84	1.00×	14.43	86.73	6.42	7.48	1.00×
	LMCache	37.54	88.19	38.71	7.57	1.30×	14.41	87.94	7.41	4.73	1.58×
	Quest	35.50	88.16	21.91	13.07	0.75×	16.73	76.62	5.96	7.00	1.07×
	PagedEviction	29.48	90.87	56.22	9.69	1.02×	13.60	76.95	9.41	6.16	1.21×
	PagePrune	37.40	72.62	73.88	4.14	**2.38**×	14.36	74.10	12.25	1.94	**3.86**×
LB	vLLM	32.29	88.16	30.65	5.63	1.00×	29.95	76.95	9.83	7.47	1.00×
	LMCache	32.29	88.19	37.86	5.93	0.95×	29.91	87.94	12.74	6.69	1.12×
	Quest	21.01	88.16	33.63	5.67	0.99×	21.58	76.62	12.11	4.08	1.83×
	PagedEviction	30.65	88.83	45.66	2.22	2.54×	24.41	76.62	14.32	3.51	2.13×
	PagePrune	32.37	76.83	59.34	1.57	**3.59**×	28.98	76.62	15.35	2.53	**2.95**×
InfB	vLLM	16.17	88.19	0.67	388.57	1.00×	31.83	87.87	0.14	382.63	1.00×
	LMCache	16.17	88.22	0.46	168.05	**2.31**×	31.83	87.91	0.46	233.11	1.64×
	Quest	14.35	78.07	0.10	834.67	0.47×	32.43	78.16	0.16	712.79	0.54×
	PagedEviction	13.15	88.19	0.48	511.58	0.76×	30.49	87.87	0.37	282.64	1.35×
	PagePrune	13.13	78.05	1.10	178.27	2.18×	30.41	75.10	0.85	141.22	**2.71**×
LB-v2	vLLM	24.00	88.19	0.36	61.01	1.00×	36.00	87.87	0.37	66.35	1.00×
	LMCache	24.00	88.22	0.88	45.51	1.34×	36.00	87.91	1.19	31.20	2.13×
	Quest	22.00	78.05	0.31	53.05	1.15×	36.00	76.63	1.03	36.20	1.83×
	PagedEviction	24.00	78.05	1.02	56.71	1.08×	36.00	76.63	0.54	46.98	1.41×
	PagePrune	24.00	76.83	1.25	20.14	**3.03**×	36.00	64.16	1.23	22.58	**2.94**×

Table 2: Vision-Language inference results on NExTQA, ActQA, and EgoSchema using Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct.

	Method	Qwen3-VL-4B-Instruct					Qwen3-VL-8B-Instruct				
		Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑	Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑
NExTQA	vLLM	82.38	78.35	0.27	7.53	1.00×	82.12	78.94	0.25	8.09	1.00×
	LMCache	82.25	62.40	1.06	1.89	3.98×	82.73	60.57	1.17	1.72	4.70×
	Quest	82.35	74.82	0.40	5.03	1.50×	82.73	79.51	0.41	4.93	1.64×
	PagedEviction	82.28	64.23	0.78	2.55	2.95×	82.80	71.11	0.99	2.02	4.01×
	PagePrune	82.25	41.49	1.22	1.64	**4.59**×	82.47	43.97	1.29	1.55	**5.22**×
ActQA	vLLM	41.25	51.25	1.28	3.23	1.00×	43.13	82.28	1.36	3.02	1.00×
	LMCache	41.25	87.78	0.62	3.23	1.00×	42.81	89.81	1.51	2.63	1.15×
	Quest	41.56	65.65	1.19	3.49	0.93×	43.13	69.83	0.87	4.42	0.68×
	PagedEviction	41.56	65.64	1.89	2.18	**1.48**×	43.44	69.83	1.48	2.60	**1.16**×
	PagePrune	40.31	51.25	1.48	2.36	1.37×	43.13	54.89	1.48	2.60	**1.16**×
EgoSchema	vLLM	73.40	78.35	0.27	7.53	1.00×	72.20	78.94	0.25	8.09	1.00×
	LMCache	73.40	82.11	0.30	6.58	1.14×	72.20	55.95	0.29	6.81	1.19×
	Quest	73.40	61.76	0.26	7.59	0.99×	72.20	66.50	0.25	8.13	0.99×
	PagedEviction	73.40	61.76	0.34	5.84	1.29×	72.20	66.49	0.31	6.49	1.25×
	PagePrune	73.20	55.39	0.40	4.95	**1.52**×	72.00	55.91	0.39	5.11	**1.58**×

working set does not necessarily translate into proportional E2E acceleration. This gap suggests that the organization of the retained KV pages plays an important role in determining the realized acceleration. By selecting important pages and compacting the retained working set into tile-aligned layouts, PagePrune jointly reduces KV access and the execution inefficiency associated with irregular page organization. The variation in speedup across workloads and model sizes further supports the need for execution-aware KV management, as the benefit of pruning depends not only on how many KV pages are removed but also on how the retained pages are presented to the attention execution.

Long-context VQA. Table 2 extends the evaluation to multimodal workloads, where the large number of visual tokens substantially increases the KV-cache footprint and makes efficient KV access increasingly important. PagePrune delivers substantial memory and E2E benefits across these workloads, with the largest gains observed on NExTQA. For Qwen3-VL-8B-Instruct, PagePrune reduces peak memory from 78.94 to 43.97 GB, while reducing latency from 8.09 to 1.55 seconds and achieving $5.22\times$ speedup with comparable accuracy. E2E gains vary across workloads because the contribution of KV-cache access to overall execution differs with the workload. In particular, PagePrune consistently reduces memory and improves latency on both ActQA and EgoSchema, while the larger gains on NExTQA indicate that PagePrune is especially effective when KV-cache access constitutes a substantial portion of the inference cost. Overall, these results demonstrate that PagePrune translates KV-cache reduction into practical E2E acceleration in multimodal inference, with the realized benefit naturally reflecting the execution characteristics of each workload.

Long-context VS. Table 3 evaluates PagePrune on ActCap and DREAM1K with Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct. Across both benchmarks and model sizes, PagePrune consistently converts KV-cache reduction into substantial E2E serving gains while preserving comparable ROUGE-L. The benefit is particularly pronounced for the larger 8B model: PagePrune achieves $7.91\times$ speedup on ActCap and $5.02\times$ on DREAM1K, reducing latency to 2.54 s and 1.40 s while reaching 93.73 and 175.49 tok/s, respectively. This scaling trend highlights the advantage of execution-aware KV working-set management. As model execution becomes more demanding, reducing the active KV

Table 3: Video-Language results on ActCap and DREAM1K using Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct.

	Method	Qwen3-VL-4B-Instruct					Qwen3-VL-8B-Instruct				
		ROUGE-L ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑	ROUGE-L ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑
ActCap	vLLM	13.62	75.71	48.78	5.01	1.00×	13.96	78.02	11.84	20.10	1.00×
	LMCache	13.62	60.93	85.98	2.84	1.76×	13.87	83.52	85.42	2.83	7.10×
	Quest	14.05	61.10	28.21	8.68	0.58×	13.96	82.18	36.16	6.58	3.05×
	PagedEviction	14.05	62.22	57.52	4.26	1.18×	13.96	82.18	61.68	3.86	5.21×
	PagePrune	13.88	45.70	115.86	2.12	**2.36**×	14.10	77.07	93.73	2.54	**7.91**×
DREAM1K	vLLM	12.70	73.67	77.36	3.18	1.00×	12.80	70.98	35.24	7.03	1.00×
	LMCache	12.63	47.07	128.88	1.89	1.68×	13.10	47.99	121.98	2.00	3.52×
	Quest	12.70	53.39	81.15	3.03	1.05×	12.80	58.53	53.73	4.61	1.52×
	PagedEviction	12.70	53.39	99.82	2.47	1.29×	12.80	58.53	108.64	2.28	3.08×
	PagePrune	12.82	41.51	188.21	1.22	**2.61**×	12.66	41.87	175.49	1.40	**5.02**×

Table 4: Component ablation of PagePrune on DREAM1K and EgoSchema with Qwen3-VL-4B-Instruct. Score denotes ROUGE-L on DREAM1K and accuracy (%) on EgoSchema. The Tile Align. and Compact variants are both built on page scoring, while PagePrune combines all three components.

	Configuration	Scoring	Tile Align.	Compact	Score ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑
DREAM1K	vLLM	×	×	×	12.70	73.67	77.36	3.18	1.00×
	+ Scoring	✓	×	×	12.70	53.39	70.90	3.48	0.91×
	+ Tile Align.	✓	✓	×	12.75	53.39	78.10	3.19	1.00×
	+ Compact	✓	×	✓	12.70	61.67	81.73	3.02	1.05×
	PagePrune	✓	✓	✓	12.82	41.51	188.21	1.22	**2.61**×
EgoSchema	vLLM	×	×	×	73.40	78.35	0.27	7.53	1.00×
	+ Scoring	✓	×	×	73.00	69.60	0.27	7.35	1.02×
	+ Tile Align.	✓	✓	×	73.00	69.60	0.30	6.77	1.11×
	+ Compact	✓	×	✓	73.20	61.76	0.30	6.59	1.14×
	PagePrune	✓	✓	✓	73.20	55.39	0.40	4.95	**1.52**×

footprint can yield increasingly substantial system-level benefits rather than merely saving memory. Even on the 4B models, PagePrune achieves $2.36\times$ and $2.61\times$ speedups on ActCap and DREAM1K, respectively, while reducing peak memory to 45.70 and 41.51 GB. These results demonstrate that PagePrune's page-level, execution-aligned design is effective across diverse video-language workloads and model scales, translating KV-cache efficiency into both memory savings and substantial E2E serving acceleration.

Across language, VQA, and video workloads, the results establish a consistent pattern: KV pruning and E2E acceleration are not equivalent objectives. Methods that only reduce the logical number of retained KV pages can leave substantial execution inefficiency in the attention backend, whereas PagePrune explicitly aligns the retained working set with the execution granularity of the underlying kernels. This execution-aware organization allows PagePrune to convert KV reduction into practical latency and throughput improvements, with the largest gains appearing when KV access constitutes a major fraction of the E2E inference cost. The results therefore support the central design principle of PagePrune: effective KV-cache pruning should optimize not only which pages are retained, but also how the retained pages are presented to the execution engine.

4.2 Ablation Studies

This section evaluates PagePrune through component ablation and sensitivity analyses. We first examine the effect of progressively incorporating its components, followed by sensitivity studies on batch size, pruning interval, tile size, and recomputation overhead. Additional analyses on the retention ratio, scoring layer, μ sweep, and sink/recent protection are provided in Appendix F.

Component-wise ablation. Table 4 separates the roles of page scoring, tile alignment, and block-table compaction. Page scoring alone substantially reduces the KV footprint, from 73.67 to 53.39 GB on DREAM1K, but does not translate into an E2E gain: throughput decreases from 77.36 to 70.90 tok/s. Tile alignment introduces a small amount of page expansion by retaining the complete execution tile around selected pages, trading a modest increase in the logical working set for a more regular access pattern at the attention-kernel granularity. This trade-off improves throughput to 78.10 tok/s on DREAM1K and reduces latency from 7.35 to 6.77 s on EgoSchema. Compact-once provides a complementary optimization by rebuilding the block-table representation after pruning, reducing the representation overhead of the retained working set and further lowering latency to 3.02 s on DREAM1K and 6.59 s on EgoSchema.

Table 5: Sensitivity to batch size on EgoSchema and ActCap.

Batch Size	Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓
EgoSchema | Qwen3-VL-8B-Instruct				
1	70.60	47.16	0.29	7.08
8	72.00	54.03	0.32	6.24
16	72.00	55.91	0.33	6.11
32	72.00	55.91	0.39	5.11
ActCap | Qwen3-VL-4B-Instruct				
1	13.93	71.80	35.59	6.91
8	14.09	62.79	91.86	2.66
16	13.64	65.95	86.62	2.79
32	13.88	45.70	115.86	2.12

Table 6: Sensitivity to the pruning application interval on EgoSchema and LongBench-v2.

Interval	Acc. (%) ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓
EgoSchema | Qwen3-VL-4B-Instruct				
8	73.20	55.39	0.28	7.05
16	73.20	55.39	0.28	7.20
32	73.20	55.39	0.31	6.46
64	73.20	55.39	0.40	4.95
128	73.20	66.79	0.39	5.09
LongBench-v2 | Qwen3-8B				
8	36.00	75.80	0.36	32.91
16	36.00	75.83	0.35	33.81
32	36.00	75.83	0.59	33.96
64	36.00	64.16	1.23	22.58
128	36.00	80.56	0.30	157.13

Combining all three components yields a substantially larger E2E gain: PagePrune reaches 188.21 tok/s and 1.22 s latency on DREAM1K, and 0.40 tok/s and 4.95 s on EgoSchema, while maintaining comparable output quality. The ablation thus highlights a division of roles: scoring identifies low-utility pages for removal, tile alignment trades limited retention expansion for kernel-compatible execution granularity, and compact-once maintains a compact block-table representation for subsequent decoding. Together, they bridge the gap between KV reduction and effective execution, achieving $2.61\times$ and $1.52\times$ speedups on DREAM1K and EgoSchema, respectively.

Batch-size sensitivity. Table 5 evaluates the scalability of PagePrune under different serving batch sizes on EgoSchema and ActCap. Across both benchmarks, PagePrune exhibits consistent behavior as serving concurrency increases, with answer accuracy remaining largely unchanged across batch sizes. On EgoSchema with Qwen3-VL-8B-Instruct, accuracy ranges from 70.60% to 72.00% as the batch size increases from 1 to 32. During the same range, throughput increases from 0.29 to 0.39 token/s, while latency decreases from 7.08 to 5.11 seconds. On ActCap with Qwen3-VL-4B-Instruct, accuracy remains within a narrow range from 13.64% to 14.09%, while throughput increases substantially from 35.59 to 115.86 token/s and latency decreases from 6.91 to 2.12 s. The stable accuracy across substantially different batch sizes suggests that PagePrune's page selection remains robust under varying levels of request concurrency rather than being tied to a particular serving configuration. The consistent efficiency improvements further demonstrate that PagePrune can naturally operate with batched serving and benefit from increased concurrency. Overall, these results demonstrate that PagePrune provides a batch-size-robust KV working-set management mechanism that remains effective across different serving scales.

Pruning interval. Table 6 studies the pruning interval G, which controls how frequently PagePrune refreshes the retained KV working set. Accuracy remains unchanged across all settings on both EgoSchema (73.20%) and LongBench-v2 (36.00%), indicating that the update frequency has little effect on task quality within the evaluated range. Increasing G from 8 to 64 substantially improves serving efficiency by amortizing page-scoring, ranking, and cache-update overhead: throughput increases from 0.28 to 0.40 tok/s on EgoSchema and from 0.36 to 1.23 tok/s on LongBench-v2, while latency decreases from 7.05 to 4.95 s and from 32.91 to 22.58 s, respectively. However, overly infrequent updates become counterproductive: at $G = 128$, EgoSchema memory rises from 55.39 to 66.79 GB with a slight latency regression to 5.09 s, while LongBench-v2 degrades more sharply to 80.56 GB and 157.13 s. These results reveal a trade-off between amortizing update overhead and refreshing the KV working set in a timely manner, with $G = 64$ providing the most favorable overall balance across the two workloads.

Tile Size. We study the sensitivity of PagePrune to the number of pages K grouped into each execution tile on both multimodal and text-only workloads. As shown in Figure 3, model quality is largely insensitive to this choice, whereas the system-level efficiency varies substantially with K. On EgoSchema, accuracy varies by only 0.2 percentage points across all settings. In contrast, throughput exhibits a clear non-monotonic trend and peaks at $K = 8$ with 0.45 tok/s, while memory decreases to 55.39 GB and remains at this level for $K \geq 8$. A similar pattern appears on LongBench: accuracy remains within 32.06–32.84, whereas $K = 8$ simultaneously achieves the lowest memory usage of 76.83 GB and the highest throughput of 59.34 tok/s. These results indicate a trade-off in execution granularity. A small K provides finer-grained page management but exposes a less compact execution layout, whereas an overly large K reduces the flexibility of page-level selection.

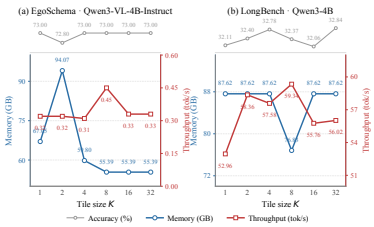

Figure 3: Tile-size sensitivity on EgoSchema with Qwen3-VL-4B-Instruct and LongBench with Qwen3-4B.

Table 7: Recomputation overhead of PagePrune after physical KV-page eviction. Recovery cost remains a small fraction of E2E latency across all evaluated workloads. VL-4B/8B denote Qwen3-VL-4B/8B-Instruct; 4B/8B denote Qwen3-4B/8B.

Benchmark	Model	Tokens	Recov. (tok/s)	Decode (tok/s)	Per Req. (s)	E2E (%)
DREAM1K	VL-4B	12,857	6,875	118.8	0.040	1.8
	VL-8B	17,017	4,514	106.4	0.080	3.2
ActCap	VL-4B	37,457	6,264	59.8	0.120	2.9
	VL-8B	24,012	4,084	77.0	0.120	3.8
LongBench	4B	543,014	92,141	29.1	0.023	0.2
	8B	708,946	75,557	27.3	0.036	0.2
LB-v2	4B	1,166,375	21,416	5.28	1.090	1.9
	8B	277,318	230,648	2.30	0.024	0.1

Across both workloads, $K = 8$ provides a favorable balance between these effects, and we therefore use it as the default setting in subsequent experiments.

Recomputation Overhead of PagePrune's Eviction. PagePrune physically evicts low-importance KV pages and recomputes their KV states when they are needed again, raising the question of whether recovery cost could offset the benefit of cache reduction. Table 7 shows that this overhead remains small across both multimodal and long-context workloads. Although the total number of recovered tokens can be large, recovery is executed at substantially higher throughput than AR decoding. As a result, the amortized recovery cost is only 0.04–0.12 s per request on DREAM1K and ActCap and 0.023–1.09 s on LongBench and LongBench-v2, corresponding to only 0.1%–3.8% of E2E latency. These results indicate that physical KV-page eviction can reduce the active cache footprint while keeping recovery overhead small relative to overall inference time.

5 Related Work

KV-cache management has evolved from static retention to adaptive and serving-aware management. Early methods such as StreamingLLM (Xiao et al., 2024b), Scissorhands (Liu et al., 2023), H2O (Zhang et al., 2023), and Keyformer (Adnan et al., 2024) retain or evict tokens based on fixed heuristics or estimated importance. SnapKV (Li et al., 2024) shifts selection to prefill, while PyramidKV (Cai et al., 2024), Ada-KV (Feng et al., 2024), and VisCache (Wang et al., 2026b) adapt KV budgets across layers, heads, or visual tokens. Orthogonal approaches reduce KV storage through quantization (Liu et al., 2024c; Hooper et al., 2024; Xiao et al., 2023) or exploit head specialization (Xiao et al., 2024a). Recent systems further consider serving and execution: Quest (Tang et al., 2024) retrieves query-relevant pages, PagedEviction (Chitty-Venkata et al., 2026) performs block-wise eviction, PackKV (Jiang et al., 2026) co-designs KV representation with GPU execution, and AsymCache (Shi et al., 2026) supports lossless eviction with non-contiguous KV processing. Despite this progress, existing methods primarily focus on what to retain, compress, or evict, while the organization of the retained KV for efficient execution remains less explored. In particular, attention backends process retained KV at different execution granularities, so arbitrary page-level selections may not map efficiently to backend execution. PagePrune addresses this pruning–execution gap by jointly considering what to retain and how to organize it: it derives temporally aggregated page utility from backend attention probabilities, aligns retention with the backend execution granularity, and compacts the resulting block-table representation, allowing KV reduction to translate into E2E gains without specialized sparse operators.

6 Conclusion

We present PagePrune, an execution-aware, page-granular KV-cache pruning policy for paged-attention serving systems. PagePrune combines dynamic attention-based page selection with execution-aware cache reorganization, reducing KV-cache memory and unnecessary decode-time KV accesses while preserving comparable task quality. Across long-context language, multimodal, and video-language workloads, PagePrune achieves up to $7.91\times$ E2E speedup over vanilla vLLM. These results show that effective KV pruning requires jointly considering both what to retain and how the retained cache is executed.

REFERENCES

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

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

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Krishna Teja Chitty-Venkata, Jie Ye, Siddhisanket Raskar, Anthony Kougkas, Xian Sun, Murali Emani, Venkatram Vishwanath, and Bogdan Nicolae. Pagedeviction: Structured block-wise kv cache pruning for efficient large language model inference. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 3207–3218, 2026.

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. *arXiv preprint arXiv:2407.11550*, 2024.

Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant Nair, and Poulami Das. Dialogue without limits: Constant-sized kv caches for extended responses in llms. *arXiv preprint arXiv:2503.00979*, 2025.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

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

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

Bo Jiang, Taolue Yang, Youyuan Liu, Xubin He, Sheng Di, and Sian Jin. Packkv: Reducing kv cache memory footprint through llm-aware lossy compression. In *2026 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 598–611. IEEE, 2026.

Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 706–715, 2017.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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, 2023.

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

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

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

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024b.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, et al. Lmcache: An efficient kv cache layer for enterprise-scale llm inference. *arXiv preprint arXiv:2510.09665*, 2025.

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. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

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

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. Egoschema: A diagnostic benchmark for very long-form video language understanding. *Advances in Neural Information Processing Systems*, 36:46212–46244, 2023.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 1133–1150, 2025.

Chunan Shi, Yilei Chen, Yilin Chen, Xupeng Miao, and Bin Cui. Multi-segment attention: Enabling efficient kv-cache management for faster large language model serving. *arXiv preprint arXiv:2606.02964*, 2026.

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

Heng Wang, Jielin Qiu, Wenting Zhao, Cheng Qian, Liangwei Yang, Jiawei Han, Heng Ji, Silvio Savarese, Shelby Heinecke, and Huan Wang. Random attention: Rethinking kv cache eviction for efficient reasoning. *arXiv preprint arXiv:2609.03430*, 2026a.

Jiawei Wang, Liping Yuan, Yuchen Zhang, and Haomiao Sun. Tarsier: Recipes for training and evaluating large video description models. *arXiv preprint arXiv:2407.00634*, 2024.

Lyuke Wang, Zhuo Li, and Guangxu Zhu. Viscache: Visual kv cache pruning for efficient vision large language model inference. *arXiv preprint arXiv:2608.24063*, 2026b.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. *URL https://arxiv. org/abs/2309.17453*, 1, 2024b.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9777–9786, 2021.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Zhou Yu, Dejing Xu, Jun Yu, Ting Yu, Zhou Zhao, Yueting Zhuang, and Dacheng Tao. Activitynet-qa: A dataset for understanding complex web videos via question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 9127–9134, 2019.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A LLM USAGE STATEMENT

We employ a large language model (LLM) as a general-purpose writing assistant to improve the clarity and fluency of the text. Its role is limited to refining linguistic expression and enhancing overall readability and coherence.

B ALGORITHM

Algorithm 1: PagePrune

Input: Logical page sequence $\mathcal{P}_r(t)$; active page set $\mathcal{C}_r(t)$; previously pruned page set $\mathcal{D}_r(t)$; stored token inputs; target retention ratio τ; smoothing factor μ; update interval G; protected page counts K_s and K_r

Output: Updated active page set $\mathcal{C}_r(t)$ and block table

if $t \bmod G \neq 0$ **then**
  return $\mathcal{C}_r(t)$ and the current block table;

`// Restore pages pruned in the previous interval`

foreach $p \in \mathcal{D}_r(t)$ **do**
  Recompute and restore the KV states of p from its stored token inputs and preceding context;
  $\mathcal{C}_r(t) \leftarrow \mathcal{C}_r(t) \cup \{p\}$;

`// Update page importance`

foreach $p \in \mathcal{C}_r(t)$ **do**
  Compute attention mass $a_p^{(t)}$;
  $\bar{A}_p^{(t)} \leftarrow \mu a_p^{(t)} + (1-\mu)\bar{A}_p^{(t-1)}$;

`// Select pages under the target budget`

$B_r(t) \leftarrow \lceil \tau N_r(t) \rceil$;

$R_r(t) \leftarrow \mathrm{Top}_{B_r(t)}(\bar{A}) \cup \{p_1, \ldots, p_{K_s}\} \cup \{p_{N_r(t)-K_r+1}, \ldots, p_{N_r(t)}\}$;

`// Align retention decisions to KV-tile boundaries`

$\tilde{R}_r(t) \leftarrow \emptyset$;

foreach $p \in \mathcal{C}_r(t)$ **do**
  if $T(p) \cap R_r(t) \neq \emptyset$ **then**
    $\tilde{R}_r(t) \leftarrow \tilde{R}_r(t) \cup \{p\}$;

`// Evict pages and update the block table`

$\mathcal{D}_r(t) \leftarrow \mathcal{C}_r(t) \setminus \tilde{R}_r(t)$;

Release physical KV blocks for pages in $\mathcal{D}_r(t)$;

Rebuild the block table over $\tilde{R}_r(t)$ in the original logical order;

$\mathcal{C}_r(t) \leftarrow \tilde{R}_r(t)$;

return $\mathcal{C}_r(t)$ and the updated block table;

C ADDITIONAL LONG-CONTEXT LLM RESULTS

To assess the generality of PagePrune beyond the Qwen3 family used in our main experiments, we further evaluate it across different model families and scales. Specifically, we consider Qwen2.5-32B-Instruct on RULER and LongBench, as well as LLaVA-NeXT-Video-7B on ActCap and ActQA. These experiments complement our main evaluation and examine whether the benefits of page-level KV management persist across different model architectures and workloads. All methods are evaluated under the same inference configuration for each benchmark.

Tables 8 and 9 demonstrate that PagePrune generalizes to a substantially larger text-only LLM. On Qwen2.5-32B-Instruct, PagePrune achieves 90.20% accuracy on RULER, slightly above the 90.14% obtained by vanilla vLLM, while reducing peak GPU memory from 93.41 to 79.62 GB and latency from 19.72 to 17.86 s. This corresponds to a $1.10\times$ latency-based relative speedup with no observable loss in retrieval accuracy. On LongBench, PagePrune similarly preserves accuracy (29.28% vs.

Table 8: Long-context inference performance on RULER using Qwen2.5-32B-Instruct. All methods are evaluated under identical decoding settings. Relative speedup is computed from E2E latency.

Method	Accuracy	Peak GPU Mem (GB)	Throughput (tok/s)	Latency (s)	Relative Speed
vLLM	90.14	93.41	28.79	19.72	1.00 ×
PagePrune	90.20	79.62	54.42	17.86	1.10 ×

Table 9: Long-context inference performance on LongBench using Qwen2.5-32B-Instruct. All methods are evaluated under identical decoding settings. Relative speedup is computed from E2E latency.

Method	Accuracy	Peak GPU Mem (GB)	Throughput (tok/s)	Latency (s)	Relative Speed
vLLM	29.26	94.15	28.62	56.41	1.00 ×
PagePrune	29.28	76.10	33.54	47.95	1.18 ×

Table 10: Performance on individual LongBench datasets using Qwen2.5-32B-Instruct. PagePrune preserves the performance of vanilla vLLM across diverse long-context tasks.

Method	Qasper	MultifieldQA-EN	HotpotQA	2WikiMQA	GovReport	MultiNews	Trec	TriviaQA	SamSum	PassageCount	Passage-Retrieval-EN	LCC	RepoBench-p	Average
vLLM	17.53	39.10	10.52	11.06	18.50	15.01	46.00	83.14	5.25	20.00	99.00	7.20	8.14	29.26
PagePrune	17.53	39.09	10.52	11.13	18.49	15.04	46.00	83.15	5.23	20.00	98.67	7.35	8.08	29.28

29.26%) and reduces latency from 56.41 to 47.95 s, corresponding to a $1.18\times$ latency-based relative speedup.

The difference between the two benchmarks is also informative. PagePrune provides a larger gain on RULER, where long-context retrieval creates a substantial KV-cache working set, while the gain on LongBench is more moderate. Nevertheless, accuracy remains essentially unchanged on both benchmarks. These results show that PagePrune is not tied to the Qwen3 architecture or multimodal workloads: its page-level KV selection and execution-aware organization remain effective on a 32B text-only model.

Table 10 provides a finer-grained view of the LongBench results. PagePrune closely matches vanilla vLLM across all evaluated tasks, with the average score changing from 29.26% to 29.28%. The largest differences are small and do not follow a consistent task-specific pattern: some tasks slightly improve, while others slightly decrease. This consistency across question answering, multi-document reasoning, summarization, retrieval, and code understanding further supports the model-agnostic nature of PagePrune's page selection mechanism.

Importantly, these results are obtained on a model family and scale that are not used in the main Qwen3-based evaluation. Together with the multimodal Qwen3-VL experiments, they indicate that PagePrune's effectiveness does not depend on a particular model family or modality, but instead arises from the general property that long-context autoregressive inference maintains a large paged KV cache whose execution cost can be reduced through importance-aware page management.

We further examine model generalization on LLaVA-NeXT-Video-7B, which differs from the Qwen3-VL models used in the main experiments. Tables 11 and 12 report results on ActCap and ActQA, respectively. On ActCap, PagePrune achieves the highest ROUGE-L of 15.63 while reducing latency from 3.78 to 2.00 s relative to vanilla vLLM. This corresponds to a $1.89\times$ latency-based relative speedup, while throughput increases from 68.6 to 113.5 tok/s. These results show that PagePrune preserves task quality while substantially improving E2E execution efficiency on a different video-language model family.

On ActQA, all methods achieve the same accuracy of 72.0%. PagePrune reduces latency from 0.62 to 0.24 s, corresponding to a $2.58\times$ latency-based relative speedup. Its throughput also increases from 9.4 to 19.0 tok/s. These results further demonstrate that PagePrune can reduce E2E latency and improve execution efficiency while preserving task quality on LLaVA-NeXT-Video-7B.

Overall, these additional experiments broaden the scope of our evaluation beyond the Qwen3/Qwen3-VL family. PagePrune is effective on Qwen2.5-32B-Instruct, a larger text LLM, and remains applicable to LLaVA-NeXT-Video-7B, a different video-language model family. The consistent preservation of task quality across RULER, LongBench, ActCap, and ActQA suggests that PagePrune

Table 11: Performance of LLaVA-NeXT-Video-7B on ActCap. Relative speedup is computed from E2E latency.

Method	ROUGE-L	Memory (GB)	Throughput (tok/s)	Latency (s)	Relative Speedup
Vanilla vLLM	15.54	74.3	68.6	3.78	1.00 ×
Quest	15.54	74.3	23.4	9.74	0.39 ×
PagedEviction	15.54	74.3	54.3	4.20	0.90 ×
PagePrune	15.63	26.0	113.5	2.00	1.89 ×

Table 12: Performance of LLaVA-NeXT-Video-7B on ActQA. Relative speedup is computed from E2E latency.

Method	Accuracy (%)	Memory (GB)	Throughput (tok/s)	Latency (s)	Relative Speedup
Vanilla vLLM	72.0	74.3	9.4	0.62	1.00 ×
Quest	72.0	74.3	11.0	0.41	1.51 ×
PagedEviction	72.0	74.3	15.8	0.29	2.14 ×
PagePrune	72.0	43.0	19.0	0.24	2.58 ×

does not rely on a particular model family or modality. Instead, its core design—importance-based page selection combined with execution-aware page organization—targets the underlying KV-cache behavior shared by long-context autoregressive inference across different model families and modalities.

D Comparison with Random Attention

To further isolate the effect of PagePrune's importance-aware page selection, we include a supplementary comparison with Random Attention (Wang et al., 2026a). We evaluate this baseline on two representative VS benchmarks, ActCap and DREAM1K, using Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct.

Table 13: Video-Language results on ActCap and DREAM1K using Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct backbone.

Method	Qwen3-VL-4B-Instruct					Qwen3-VL-8B-Instruct				
	ROUGE-L ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑	ROUGE-L ↑	Mem. (GB) ↓	Thp. (tok/s) ↑	Lat. (s) ↓	RS ↑
ActCap										
vLLM	13.62	75.71	48.78	5.01	1.00×	13.96	78.02	11.84	20.10	1.00×
Random Attention	13.69	65.54	51.06	4.76	1.05×	14.08	89.96	39.80	6.05	3.32×
PagePrune	13.88	45.70	115.86	2.12	**2.36**×	14.10	77.07	93.73	2.54	**7.91**×
DREAM1K										
vLLM	12.70	73.67	77.36	3.18	1.00×	12.80	70.98	35.24	7.03	1.00×
Random Attention	12.80	82.10	79.37	3.09	1.03×	12.82	81.80	66.30	3.69	1.90×
PagePrune	12.82	41.51	188.21	1.22	**2.61**×	12.66	41.87	175.49	1.40	**5.02**×

Table 13 shows that Random Attention can preserve accuracy reasonably well, with ROUGE-L remaining close to the vLLM baseline on both benchmarks. However, its E2E benefits are substantially smaller than those of PagePrune. On ActCap with Qwen3-VL-4B-Instruct, Random Attention achieves only $1.05\times$ speedup, compared with $2.36\times$ for PagePrune; with Qwen3-VL-8B-Instruct, the gap increases to $3.32\times$ versus $7.91\times$. A similar trend appears on DREAM1K, where PagePrune achieves $2.61\times$ and $5.02\times$ speedup, compared with only $1.03\times$ and $1.90\times$ for Random Attention.

These results suggest that reducing the number of retained pages alone is insufficient to consistently translate KV-cache reduction into E2E acceleration. PagePrune's attention-based ranking preferentially preserves useful pages while producing a substantially smaller and more execution-efficient working set.

E Inference Stability

Table 14 reports the mean and standard deviation across three independent runs under identical configurations. System-level metrics such as peak GPU memory, throughput, and latency are

Table 14: Run-to-run stability of the core comparison (3 independent runs, mean ± std). Quality metrics, ROUGE-L on DREAM1K and LongBench and accuracy on EgoSchema and LongBench-v2, are near-deterministic under greedy decoding, while efficiency metrics exhibit bounded run-to-run variance.

	Score ↑	Mem (GB) ↓	Throughput (Tok/s) ↑	Latency (s) ↓
DREAM1K				
vLLM	12.70 ± 0.00	73.67 ± 0.00	77.4 ± 19.1	3.18 ± 0.40
PagePrune	12.82 ± 0.18	41.51 ± 0.00	188.2 ± 4.6	1.22 ± 0.06
EgoSchema				
vLLM	73.40 ± 0.00	78.35 ± 0.00	0.27 ± 0.02	7.53 ± 2.1
PagePrune	73.20 ± 0.00	55.39 ± 0.00	0.40 ± 0.01	4.95 ± 0.18
LongBench				
vLLM	29.95 ± 0.00	76.95 ± 0.43	9.83 ± 2.4	7.47 ± 0.10
PagePrune	28.98 ± 0.34	76.62 ± 0.00	15.35 ± 0.8	2.53 ± 0.84
LongBench-v2				
vLLM	36.00 ± 0.00	87.87 ± 0.42	0.37 ± 0.00	66.35 ± 0.13
PagePrune	36.00 ± 0.00	64.16 ± 0.00	1.23 ± 0.00	22.58 ± 0.01

sensitive to batching order, memory-allocator state, and kernel scheduling, all of which may vary across runs; quality metrics under greedy decoding, in contrast, are expected to be near-deterministic. To reflect robust system behavior rather than a single favorable run, we repeat the core comparison three times with identical configuration and report the mean and standard deviation across three independent runs.

We evaluate stability on four representative benchmarks: DREAM1K and EgoSchema with the Qwen3-VL-4B-Instruct backbone, and LongBench and LongBench-v2 with the Qwen3-8B-Instruct backbone. The former pair covers multimodal long-video and egocentric video understanding, while the latter pair covers long-context language tasks. Each of the three runs launches a fresh engine with the same workload; no random seed is fixed, so batch scheduling and memory layout vary naturally between runs.

Generation quality is essentially deterministic. Generation quality is highly stable across all configurations. Across the three runs, ROUGE-L and accuracy vary by at most 0.34 points, with relative standard deviations below 1.5% across all four benchmarks. Under greedy decoding with fixed inputs, the decoding trajectory is expected to remain highly consistent, with small residual variation arising from numerical non-determinism such as non-associative reductions inside the attention kernels. These results confirm that the quality measurements are stable across independent runs rather than being driven by run-to-run sampling variation. The identical accuracy on LongBench-v2, 36.00 for both systems, further shows that PagePrune introduces no measurable quality degradation on this long-context language workload.

Efficiency metrics exhibit workload-dependent variance. Peak GPU memory is highly stable across runs, with relative standard deviation below 0.6% for all configurations. Throughput and latency show larger variation, particularly on multimodal workloads and shorter requests where execution time is more sensitive to scheduling and workload composition. Nevertheless, the main performance differences remain substantial. On Qwen3-VL-4B-Instruct, PagePrune reduces peak GPU memory by 32.16 GB on DREAM1K and 22.96 GB on EgoSchema, while improving DREAM1K throughput from 77.4 to 188.2 tok/s. On Qwen3-8B-Instruct, PagePrune reduces LongBench latency from 7.47 to 2.53 seconds and LongBench-v2 latency from 66.35 to 22.58 seconds. It also reduces LongBench-v2 peak GPU memory by 23.71 GB and increases throughput from 0.37 to 1.23 tok/s, corresponding to a $3.3\times$ throughput improvement. For LongBench, PagePrune increases mean throughput from 9.83 to 15.35 tok/s while reducing latency from 7.47 to 2.53 seconds. The observed run-to-run variation therefore does not alter the overall performance trends across the evaluated workloads.

Table 15: Ablation on the KV retention ratio τ on ActQA with Qwen3-VL-8B-Instruct. τ is the target fraction of KV pages retained after pruning. KV Mem denotes the GPU memory footprint of the KV cache, while KV Ret denotes the actual fraction of KV pages retained after pruning.

Methods	τ	Acc (%)	KV Mem (GB)	KV Ret (%)	Throughput (Tok/s)	Latency (s)	Relative Speed
vLLM	1.0	43.13	1.52	100.00	1.36	3.02	1.00 ×
PagePrune	0.9	42.50	1.35	91.83	1.28	2.91	1.04 ×
PagePrune	0.8	42.50	1.18	81.96	1.46	2.79	1.08 ×
PagePrune	0.7	42.50	1.03	71.68	1.56	2.60	1.16 ×
PagePrune	0.6	42.50	0.87	62.20	1.22	2.95	1.02 ×
PagePrune	0.5	42.50	0.72	51.93	1.26	2.91	1.04 ×
PagePrune	0.4	42.19	0.57	41.93	1.14	2.97	1.02 ×
PagePrune	0.3	43.13	0.43	33.36	1.48	2.67	1.13 ×
PagePrune	0.2	42.50	0.28	18.69	1.42	2.97	1.02 ×
PagePrune	0.1	42.50	0.14	9.26	1.47	2.79	1.08 ×

Overall, these results demonstrate that the memory reduction, latency improvements, and throughput gains of PagePrune are reproducible across independent runs. Although efficiency metrics exhibit workload-dependent variance, the main conclusions remain consistent across both multimodal and long-context language settings and are not artifacts of a single execution.

F Supplementary Ablation Studies

We present supplementary ablation studies to examine the sensitivity of PagePrune to its main serving and pruning configurations. We evaluate the retention ratio, attention scoring layer, μ smoothing coefficient, sink and recent protection.

Retention-ratio sweep. Table 15 evaluates the effect of the target retention ratio τ on ActQA with Qwen3-VL-8B-Instruct. We sweep τ from 0.9 to 0.1 while keeping the other PagePrune configurations fixed. The measured KV retention decreases from 91.83% to 9.26%, closely following the target budget and demonstrating that PagePrune provides effective and controllable KV-cache reduction.

Accuracy remains stable across the entire sweep, varying only from 42.19% to 42.50%, compared with 43.13% for vanilla vLLM. Even under the most aggressive setting, $\tau = 0.1$, PagePrune retains only 9.26% of the KV pages while maintaining 42.50% accuracy. This demonstrates that PagePrune can substantially reduce the KV-cache footprint without requiring a narrow operating point for task quality.

E2E performance is also robust across a wide range of retention ratios. PagePrune achieves 1.02–$1.16\times$ relative speed across all settings, with the best performance at $\tau = 0.7$ ($1.16\times$). Notably, even highly aggressive pruning remains competitive with vanilla vLLM: $\tau = 0.3$ achieves $1.13\times$, while $\tau = 0.1$ achieves $1.08\times$ despite retaining only 9.26% of the KV pages. The absence of a monotonic relationship between KV retention and latency further shows that E2E performance is not determined solely by the amount of retained KV cache. Instead, PagePrune maintains efficient execution across a broad range of pruning budgets by aligning the retained pages with the attention execution layout. Overall, the sweep demonstrates that PagePrune offers a flexible retention–efficiency trade-off, allowing substantial KV-cache reduction while remaining robust in both task quality and E2E performance.

Final layer Attention. Table 16 studies the effect of the attention layer on DREAM1K with Qwen3-VL-4B-Instruct backbone. We compare three intermediate layers with the final decoder layer under the same PagePrune configuration. All pruning variants substantially reduce peak GPU memory from 73.7 GB for vanilla vLLM to 41.5 GB, while their effects on quality and execution efficiency differ across transformer layers.

The final layer attention provides the strongest overall result. It achieves 12.82 ROUGE-L, exceeding vLLM at 12.70, while retaining 58.5% of the KV cache. It also achieves the highest throughput of 188.21 tok/s and the lowest latency of 1.22 s among all evaluated PagePrune variants. This corresponds to a $2.61\times$ latency speedup over vanilla vLLM. In comparison, scoring at layers 8, 17,

Table 16: Final layer attention scoring ablation on DREAM1K with Qwen3-VL-4B-Instruct.

Method	ROUGE-L	Memory (GB)	Throughput (tok/s)	Latency (s)	Relative Speedup
vLLM	12.70	73.7	77.36	3.18	1.00 ×
PagePrune (Layer 8)	12.08	41.5	64.66	4.09	0.78 ×
PagePrune (Layer 17)	12.05	41.5	64.17	4.41	0.72 ×
PagePrune (Layer 26)	11.57	41.5	61.18	3.99	0.80 ×
PagePrune (Final Layer)	12.82	41.5	188.21	1.22	2.61 ×

Table 17: μ smoothing sweep on DREAM1K with Qwen3-VL-4B-Instruct backbone. μ controls how much the reuse score is smoothed toward its historical mean. Caption quality is stable across μ (12.12–12.82 ROUGE-L) while throughput holds at 155–188 tok/s.

μ	ROUGE-L	Memory (GB)	Throughput (tok/s)	Latency (s)
0.2	12.12	41.5	162.0	1.53
0.5	12.82	41.5	188.2	1.22
0.8	12.12	41.5	154.9	1.59

and 26 achieve 12.08, 12.05, and 11.57 ROUGE-L, respectively, with lower throughput and higher latency than the final-layer configuration. Their latency speedups over vLLM are 0.78×, 0.72×, and 0.80×, respectively, all below 1.00×.

These results support the use of the final decoder layer as the default attention scoring in PagePrune. More importantly, the final layer configuration provides the best execution efficiency in this ablation while maintaining the 58.5% KV retention reported in the caption, suggesting that the final layer attention signal is particularly effective for identifying pages that can be safely removed in this workload.

μ smoothing. Table 17 evaluates the coefficient μ on DREAM1K with Qwen3-VL-4B-Instruct. The choice of μ has little effect on peak memory, which remains at 41.5 GB across all settings, but has a noticeable effect on both output quality and execution efficiency. With $\mu = 0.2$, PagePrune achieves 12.12 ROUGE-L and 162.0 tok/s, while $\mu = 0.8$ yields the same ROUGE-L but lower throughput at 154.9 tok/s. The intermediate value $\mu = 0.5$ performs best, achieving 12.82 ROUGE-L, 188.2 tok/s throughput, and 1.22,s latency.

The results suggest that moderate temporal smoothing provides a favorable balance between responsiveness to recent attention observations and stability of the accumulated importance scores. In particular, $\mu = 0.5$ improves throughput by 16.2% over $\mu = 0.2$ and by 21.5% over $\mu = 0.8$, while also achieving the highest ROUGE-L among the evaluated settings. We therefore use $\mu = 0.5$ as the default coefficient.

Sink and recent protection. Figure 4 ablates the two structural safeguards of PagePrune, the preserved first page and the preserved last two pages, on the LongBench summarization benchmarks MultiNews and GovReport with Qwen3-4B and Qwen3-8B. These tasks require long generation, so page eviction fires during decoding and the safeguards have a measurable effect on output quality.

The full policy preserves both the first page and the last two pages and reaches 13.27 to 14.30 ROUGE-L on Qwen3-4B and 14.06 ROUGE-L on Qwen3-8B. Dropping the first page while keeping the last two pages lowers ROUGE-L to 6.63 on MultiNews, 5.83 on GovReport, and 8.90 on MultiNews with Qwen3-8B, a relative loss of 37% to 59% against the full policy. Dropping the last two pages while keeping the first page lowers ROUGE-L to 7.52, 7.78, and 6.77, a relative loss of 43% to 52%. Dropping both safeguards gives the lowest scores, 5.19, 2.28, and 3.33.

The degradation is consistent across both benchmarks and both model scales, and the ordering is monotone: the full policy ranks above either single safeguard, which in turn ranks above neither. The first page matters because the opening of a long document carries task defining content such as the query or the document header, and the last two pages matter because summarization attends strongly to the most recently produced context. Neither safeguard can be removed without a large quality loss, so both are kept in all main experiments.

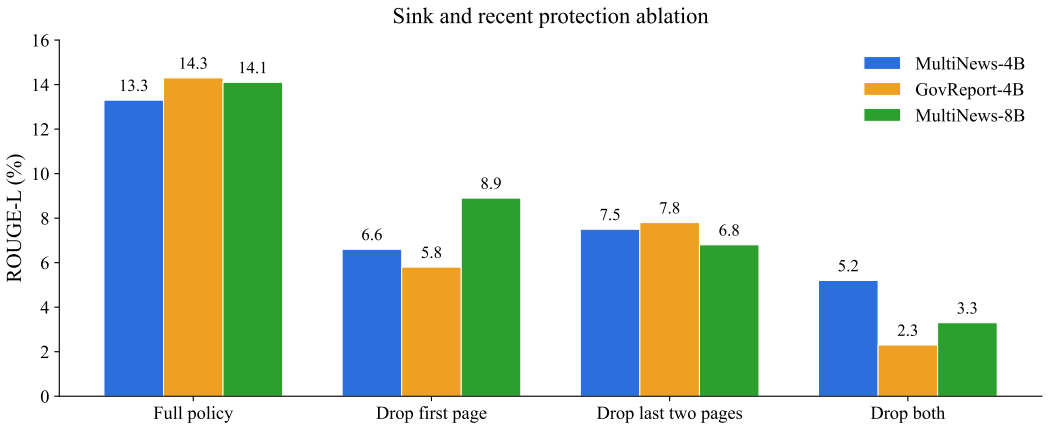

Figure 4: Sink and recent protection ablation on LongBench summarization with Qwen3-4B and Qwen3-8B under the default PagePrune configuration. The full policy preserves the first page and the last two pages, matching the main experiments. ROUGE-L is reported. Removing the first page or the last two pages degrades quality by 37% to 59% relative to the full policy, and removing both degrades it further, confirming that both safeguards are necessary.

G Needle-in-a-Haystack Retrieval Maps

The needle-in-a-haystack (NIAH) probe asks the LLM to recover a single fact planted at a controlled position inside an otherwise irrelevant haystack. Sweeping both axes provides a direct test of whether the pages containing the relevant information remain available: if the page carrying the needle is removed, the corresponding cell fails immediately rather than being averaged away in an aggregate score.

We follow the standard generator layout: haystack lengths of 1–32K tokens crossed with needle depths from 0 to 100% of the document in steps of 5%, giving 126 cells, with 2 prompts per cell (252 prompts per method). The maps show the full length–depth grid. A 64K/128K extension appears in Figure 7.

Vanilla vLLM achieves 100% accuracy on all 126 cells, providing a reference for the pruning methods. PagePrune achieves 96.0% and 90.9% mean accuracy on Qwen3-4B and Qwen3-8B, respectively, while retaining only about half of the KV pages. The degradation on 8B is real but highly localized: 121 of 126 cells on 4B and 114 of 126 cells on 8B remain perfect, and the remaining failures occur in only a small number of specific cells. These failures are all-or-nothing rather than gradual, with no clear monotonic degradation as either context length or needle depth increases.

The localized failures are consistent with the page-selection nature of PagePrune. The first and most recent pages are explicitly protected, whereas pages containing needles in the middle of the context remain subject to importance-based selection. At a small number of length–depth configurations, the page containing the needle can therefore fall outside the retained set before the answer is generated. This limitation is most visible on Qwen3-8B, where 12 of the 126 cells fail completely, and explains the gap from the 100% vanilla-vLLM reference without suggesting a systematic loss of retrieval capability.

The two baselines exhibit substantially broader degradation. PagedEviction achieves only 43.3% and 45.2% on Qwen3-4B and Qwen3-8B, respectively: only 46/52 cells remain perfect, while 63/64 cells are completely lost, with failures distributed across context lengths. Quest achieves 25.8% and 26.2%, with 87 cells fully lost on both backbones. Compared with these baselines, PagePrune preserves retrieval across nearly the entire grid at a similar $\approx 50\%$ physical KV retention.

The longer-context extension further shows that this retrieval behaviour persists at 64K and 128K contexts. PagePrune achieves 97.6% and 98.8% on Qwen3-4B and Qwen3-8B, respectively, while retaining about half of the KV pages. Although the absolute values are not directly comparable with the 1K–32K results because the filler distribution is regenerated, the maps show that PagePrune continues to preserve retrieval across the tested needle positions at substantially longer contexts.

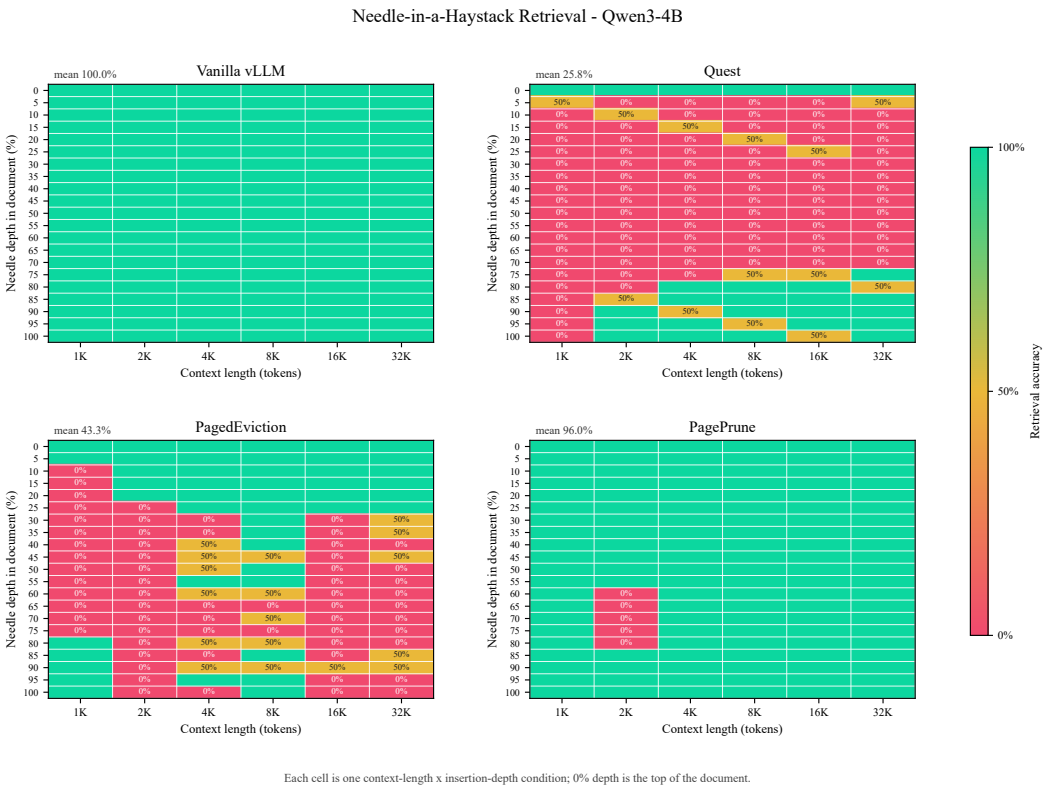

Figure 5: Needle-in-a-haystack retrieval accuracy on Qwen3-4B. Rows are the needle depth (0–100% of the document), columns are the haystack length (1–32K tokens); each cell corresponds to one context-length–insertion-depth condition and reports retrieval accuracy over two prompts. Green is retrieved, red is lost. Vanilla vLLM is green everywhere; PagePrune (96.0% mean) loses only a handful of cells, whereas PagedEviction (43.3%) and Quest (25.8%) lose cells at every context length.

Overall, the NIAH maps show that PagePrune's retrieval degradation is limited to a small set of localized cases rather than a broad failure across context lengths or needle positions. Together with the RULER and LongBench results, these results indicate that PagePrune can substantially reduce the active KV-cache footprint while maintaining high retrieval accuracy across long-context workloads.



Figure 6: Needle-in-a-haystack retrieval accuracy on Qwen3-8B, same layout as Figure 5. Vanilla vLLM remains perfect; PagePrune retains 90.9% mean accuracy with 114 of 126 cells still perfect, while PagedEviction (45.2%) and Quest (26.2%) degrade at all context lengths.



Figure 7: Needle-in-a-haystack retrieval accuracy at **64K and 128K**, extending Figures 5–6. Rows are needle depth (0–100% of the document), columns are the two long contexts, and each cell averages 50 prompts; green is retrieved, red is lost. **Top row, Qwen3-4B:** (a) Vanilla vLLM, mean 100.0%; (b) Quest, 88.1%; (c) PagedEviction, 38.1%; (d) PagePrune, 97.6%. **Bottom row, Qwen3-8B:** (e) Vanilla vLLM, 100.0%; (f) Quest, 88.1%; (g) PagedEviction, 39.3%; (h) PagePrune, 98.8%. PagePrune stays above 97% at both scales while keeping only about half of the KV pages, whereas PagedEviction loses roughly three fifths of the grid, so the ordering of the 1K–32K maps persists an order of magnitude beyond them. This long-context panel uses a freshly generated filler distribution, so its absolute numbers are not directly comparable with Figures 5–6.