GROWPAGE: ON-DEMAND KV BUDGETING FOR EFFICIENT LLM REASONING SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-output reasoning has made the key–value (KV) cache a critical memory bottleneck for efficient LLM serving. Existing KV compression methods usually rely on a predefined per-request budget and adjust only which KV states are retained, leaving the total capacity fixed throughout decoding. However, reasoning workloads exhibit substantial demand variation: different requests require different KV capacities, and the attention demand of an individual request evolves during generation. We introduce **GrowPage**, an on-demand KV budgeting framework that treats KV capacity as a runtime resource. GrowPage maintains lightweight dual-timescale query summaries to capture recent and long-term attention behaviors, and uses their relative attention working sets to estimate demand evolution. At each capacity boundary, GrowPage either compresses KV states within the current allocation or acquires an additional physical page when broader demand emerges. By integrating with PagedAttention's page-level memory abstraction, GrowPage preserves continuous batching and prefix caching. Experiments on reasoning benchmarks across multiple models show that GrowPage achieves a superior performance–throughput trade-off over existing approaches.

1 INTRODUCTION

Large reasoning models (Yuan et al., 2025; Guo et al., 2025) achieve strong performance on challenging tasks such as mathematical reasoning and code generation by producing long chains of thought (CoTs) that iteratively explore, verify, and revise candidate solutions (Wei et al., 2022). This capability, however, comes with substantial serving cost. Reasoning models can generate thousands or even tens of thousands of tokens, continuously expanding their key–value (KV) caches. When multiple long-running reasoning requests are served together, the resulting memory footprint limits the number of requests that can remain resident on GPU, making the KV cache a major bottleneck for efficient LLM serving. (Hu et al., 2025; Chitty-Venkata et al., 2026).

This pressure has motivated extensive research on KV cache compression. Early methods mainly reduce prompt KV states (Li et al., 2024; Cai et al., 2024; Feng et al., 2026), while recent approaches extend token eviction to decoding to bound cache growth during long-output generation (Ghadia et al., 2025; Cai et al., 2026; Liao et al., 2025; Ramachandran et al., 2026). However, many such methods are difficult to integrate with modern serving mechanisms such as continuous batching and prefix caching, so their algorithmic memory savings do not necessarily translate into higher serving efficiency. Zipage (Liao et al., 2026) bridges this gap through Compressed PagedAttention, combining token-wise KV eviction with PagedAttention while preserving key serving capabilities.

Yet Zipage, like most existing KV compression methods, still operates under a predefined per-request KV capacity bound. This underutilizes PagedAttention's block-based memory management, which naturally supports incremental physical-page allocation, and mismatches the heterogeneous memory demands of reasoning requests. Existing adaptive methods may change which KV states are retained or how a given budget is distributed, but typically leave the total per-request capacity fixed. GrowPage instead makes this capacity itself an online decision variable. As illustrated in Figure 1(a), a generous budget over-provisions low-demand requests, whereas an aggressive budget can under-provision demanding requests and evict reasoning-critical history. Tuning a fixed budget therefore merely shifts the operating point between wasted memory and insufficient capacity.

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.

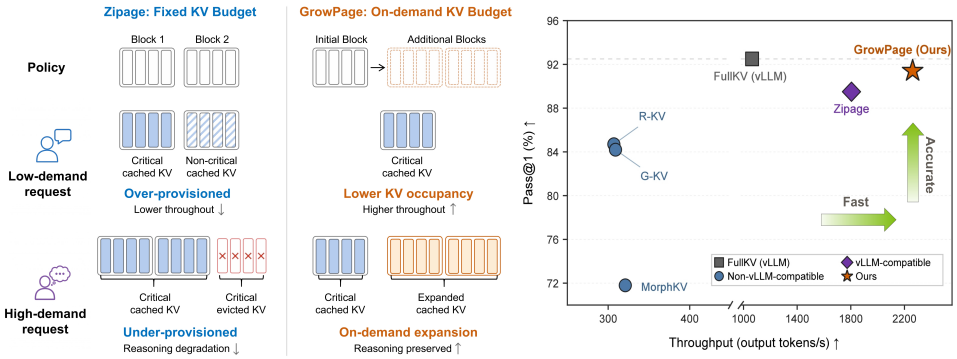


Figure 1: Illustrative comparison of KV budgeting strategies for LLM reasoning. (a) Fixed-budget Zipage and on-demand GrowPage for low- and high-demand requests. (b) Reasoning performance vs. throughput on AMC23 with Qwen3-8B.

Our analyses in Section 2 reveal that this mismatch arises at two levels: the minimum sufficient KV budget varies substantially across requests, and attention concentration within an individual request also evolves throughout decoding. These observations motivate treating per-request KV capacity as a runtime quantity rather than a static budget chosen in advance. We theoretically connect attention concentration to the KV capacity required to preserve attention outputs, and empirically show that the relative working-set difference induced by recent and long-term query summaries provides a lightweight signal for tracking demand evolution.

Based on these insights, we introduce **GrowPage**, an on-demand KV budgeting framework for efficient LLM reasoning serving. GrowPage maintains lightweight dual-timescale query summaries, using the long-timescale summary as a historical reference and the short-timescale summary to capture recent attention behavior. Their relative working sets estimate the current demand trend. At each capacity boundary, GrowPage either compresses historical KV states within the existing allocation or acquires an additional physical page. By coupling this decision with PagedAttention memory management, GrowPage adapts per-request KV capacity while preserving continuous batching and prefix caching. Across multiple mathematical reasoning and code-generation benchmarks and diverse model architectures, GrowPage consistently improves the reasoning performance–throughput trade-off; Figure 1(b) shows a representative result on AMC23 with Qwen3-8B.

Our contributions are summarized as follows:

- **We identify the demand mismatch underlying fixed KV budgeting in reasoning serving.** KV requirements vary both across requests and throughout decoding, making static capacity bounds prone to either memory over-provisioning or insufficient retention.
- **We propose GrowPage, an on-demand KV budgeting framework for reasoning LLMs.** GrowPage uses dual-timescale query summaries to estimate evolving attention demand and guide online compression or incremental physical-page allocation.
- **We co-design on-demand KV budgeting with PagedAttention-based serving.** GrowPage translates online demand estimates into page-level memory actions while preserving continuous batching and prefix caching, yielding a better reasoning performance–throughput trade-off under LLM reasoning serving.

2 Why On-Demand KV Budgeting?

Static KV budgeting assumes that memory demand can be adequately captured by a capacity fixed in advance for the entire decoding process. We examine this assumption from two complementary perspectives: inter-request heterogeneity and intra-request temporal variation.

Observation 1: Minimum sufficient KV budgets vary substantially across requests. We first quantify the KV capacity required by individual reasoning requests. For each request–budget pair, we perform multiple independent generations and aggregate their correctness to reduce sampling variance. Given a set of budgets $\mathcal{B}$, we define the *minimum sufficient KV budget* of request i as:

$$B_i^\star = \min \{B \in \mathcal{B} \mid c_i(B') = 1,\ \forall B' \in \mathcal{B}, B' \geq B\}, \tag{1}$$

where $c_i(B)$ indicates whether request i is answered correctly under budget B, as determined from multiple independent generations. Requiring correctness to hold under all larger tested budgets reduces the influence of occasional non-monotonic outcomes. Requests that are not consistently solved under any tested budget are categorized as *Unsolved*.

As shown in Figure 2, $B_i^\star$ spans the entire tested range across AIME24, AMC23, and LiveCode. This wide distribution indicates substantial inter-request heterogeneity: a small fixed budget under-provisions demanding requests, whereas a large one unnecessarily reserves KV memory for requests requiring much less capacity. Appendix I further shows that GrowPage generally allocates larger adaptive capacities to requests with higher minimum sufficient budgets, validating its request-level demand adaptation. This inter-request heterogeneity already challenges the use of a single fixed budget. However, the mismatch is not limited to differences across requests. KV demand can also evolve over time within the same request.

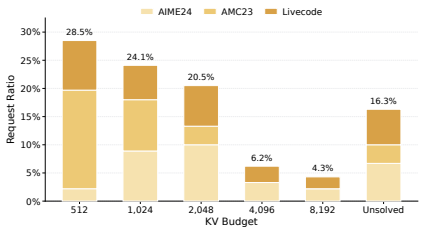


Figure 2: Distribution of minimum sufficient KV budgets for Qwen3 8B.

Observation 2: KV demand varies substantially during decoding. Prior work has shown that long-form reasoning traverses distinct reasoning stages with substantially different attention sparsity patterns (Ramachandran et al., 2026).

Consistent with this observation, Figure 3 shows that the attention sparsity of a representative request changes throughout decoding, repeatedly transitioning between concentrated and diffuse regimes. Such temporal variation suggests that the amount of historical KV information required by a request is itself non-stationary, making a capacity fixed before decoding unnecessarily conservative at some stages yet potentially insufficient at others.

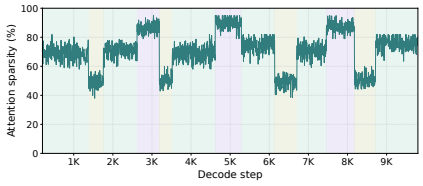


Figure 3: Attention sparsity throughout decoding for DeepSeek-R1 Distill Llama 8B on LiveCodeBench.

Together, these observations reveal that KV demand varies both across requests and over the course of decoding. They motivate treating KV capacity as an on-demand resource rather than a fixed reservation. In the next section, we formalize the connection between attention concentration and the KV capacity required to preserve attention outputs, providing the theoretical basis for GrowPage's online capacity control.

3 Methodology

3.1 Theoretical Analysis

Section 2 shows that attention concentration varies substantially throughout decoding. We now formalize why such variation implies changing KV capacity requirements. At decoding step t, let $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,N_t})$ denote the normalized attention distribution over N_t historical KV states, with $a_{t,(1)} \geq \cdots \geq a_{t,(N_t)}$ denoting the sorted attention weights. For a target coverage $p \in (0,1)$, we define the *p-coverage attention demand* as:

$$D_p(\mathbf{a}_t) = \min\left\{ k : \sum_{i=1}^{k} a_{t,(i)} \geq p \right\}. \tag{2}$$

Intuitively, concentrated attention yields a smaller D_p, whereas diffuse attention requires a larger working set. We next connect this quantity to attention-output approximation. Let $\mathbf{o}_t$ denote the full-cache attention output and $\widehat{\mathbf{o}}_t(S)$ the output obtained by retaining and renormalizing a subset S:

$$\mathbf{o}_t = \sum_{i=1}^{N_t} a_{t,i}\mathbf{v}_i, \qquad \widehat{\mathbf{o}}_t(S) = \sum_{i\in S} \frac{a_{t,i}}{m_t(S)}\mathbf{v}_i, \quad m_t(S) = \sum_{i\in S} a_{t,i}. \tag{3}$$

Under the bounded-value assumption $\|\mathbf{v}_i\|_2 \leq V_{\max}$, we define the *worst-case KV capacity*:

$$C_\delta^{\mathrm{wc}}(\mathbf{a}_t) = \min_S \left\{ |S| : \sup_{\|\mathbf{v}_i\|_2 \leq V_{\max}} \|\mathbf{o}_t - \widehat{\mathbf{o}}_t(S)\|_2 \leq \delta \right\}. \tag{4}$$

For any retained subset S, its worst-case approximation error is exactly:

$$\sup_{\|\mathbf{v}_i\|_2 \le V_{\max}} \|\mathbf{o}_t - \widehat{\mathbf{o}}_t(S)\|_2 = 2V_{\max}\big(1 - m_t(S)\big). \tag{5}$$

The upper bound follows from the triangle inequality, and is tight by assigning retained and discarded values to opposite directions with norm $V_{\max}$; the full proof is provided in Appendix C.

Theorem 3.1. *Assume $\|\mathbf{v}_i\|_2 \le V_{\max}$ for all i. For any approximation tolerance $0 < \delta < 2V_{\max}$, the minimum worst-case KV capacity satisfies $C_\delta^{\mathrm{wc}}(\mathbf{a}_t) = D_{1-\delta/(2V_{\max})}(\mathbf{a}_t)$.*

Theorem 3.1 follows because guaranteeing error at most δ is equivalent to retaining attention mass $m_t(S) \ge 1 - \delta/(2V_{\max})$, and the maximum mass achievable with k retained states is obtained by selecting the top-k attention weights. Thus, under the same tolerance, diffuse attention requires a larger robust KV capacity, whereas concentrated attention can be preserved with fewer states. For the realized value vectors of a particular request, the required capacity may be smaller; Theorem 3.1 characterizes the worst-case requirement determined solely by the attention distribution.

Combined with the non-stationary attention behavior observed in Section 2, this result motivates adapting KV capacity as attention demand evolves during decoding. We next describe how GrowPage tracks such changes online and translates them into on-demand capacity control.

3.2 Online Demand Estimation

Figure 4 illustrates the overall workflow of GrowPage. During decoding, each request maintains short- and long-timescale query summaries. When the current KV capacity is exhausted, GrowPage compares their induced historical attention working sets to estimate the demand trend and decide whether to compress the existing KV states or acquire an additional physical page.

Dual-timescale query summaries. GrowPage maintains two exponentially smoothed query representations for each request and attention layer. For clarity, we omit layer and head indices in Eq. 6. Let $\mathbf{q}_t$ denote the current query after query normalization but before RoPE:

$$\bar{\mathbf{q}}_t^x = \beta_x \bar{\mathbf{q}}_{t-1}^x + (1 - \beta_x)\mathbf{q}_t, \qquad x \in \{S, L\}, \tag{6}$$

where $\beta_S < \beta_L$; we use $\beta_S = 0.9$ and $\beta_L = 0.999$ by default. The long-timescale summary serves as a historical reference, while the short-timescale summary captures recent attention behavior. Their relative behavior therefore provides a lightweight estimate of demand evolution. Both summaries are maintained in the pre-RoPE space to avoid averaging queries under different rotary transformations. At each decision point, we calibrate their per-head RMS scales and apply RoPE at the current position before attention estimation.

Attention-demand trend. At a capacity boundary, GrowPage constructs a historical candidate set $\mathcal{C}_t$ and excludes the most recent KV block to reduce local-attention dominance. For layer ℓ and KV head h, the aligned short- and long-timescale queries induce normalized attention distributions $\mathbf{a}_{t,\ell,h}^S$ and $\mathbf{a}_{t,\ell,h}^L$ over $\mathcal{C}_t$. Following Section 3.1, we compute

$$K_{t,\ell,h}^x = \min\left\{ k : \sum_{i=1}^{k} a_{t,\ell,h,(i)}^x \ge p \right\}, \qquad r_{t,\ell,h}^x = \frac{K_{t,\ell,h}^x}{|\mathcal{C}_t|}, \quad x \in \{S, L\}, \tag{7}$$

where $a_{t,\ell,h,(i)}^x$ denotes the attention weights sorted in descending order, and we set $p = 0.99$ by default. Since page growth is a request-level decision, we follow the mean-aggregation convention of prior KV-cache methods and aggregate the working-set ratios across layers and KV heads:

$$r_t^x = \frac{1}{\sum_{\ell \in \mathcal{L}} |\mathcal{H}_\ell|} \sum_{\ell \in \mathcal{L}} \sum_{h \in \mathcal{H}_\ell} r_{t,\ell,h}^x, \qquad \Delta_t = r_t^S - r_t^L, \tag{8}$$

where $\mathcal{L}$ denotes the participating layers and $\mathcal{H}_\ell$ the KV heads in layer ℓ. A negative Δ_t indicates increasing attention concentration, whereas a positive value indicates an expanding working set. To translate this trend into a page-level decision, GrowPage compares Δ_t with a demand threshold τ: it selects *Grow by One Page* when $\Delta_t > \tau$, and otherwise performs *Compress & Hold*. We use $\tau = 0$ by default; its sensitivity is analyzed in Appendix D.1.

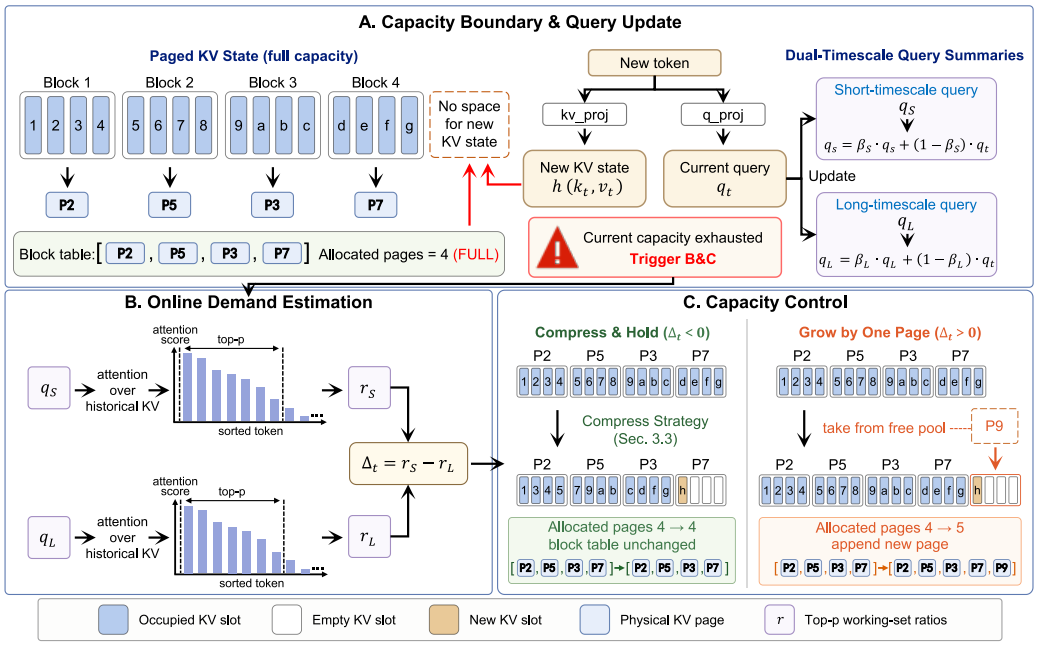


Figure 4: Overview of GrowPage. At each KV capacity boundary, GrowPage estimates the request's attention-demand trend and dynamically chooses between *Compress & Hold* and *Grow by One Page*.

3.3 Trend-Guided Capacity Control

GrowPage invokes capacity control only when the current KV allocation is exhausted. At each boundary, it chooses between *Compress & Hold* and *Grow by One Page*, naturally aligning adaptation with PagedAttention's page granularity. GrowPage adopts monotonic capacity growth rather than actively shrinking live requests, since page reclamation requires additional irreversible KV eviction; bidirectional resizing is studied in Appendix H.1.

Compress & Hold. When recent demand becomes more concentrated, GrowPage retains the current allocation and creates space by compacting historical KV states. Unlike the request-level demand signal, KV selection is performed independently for each layer and KV head. For candidate token i at layer ℓ and head h, we combine its importance under the two temporal summaries as:

$$s_{i,\ell,h} = \max\left(a^{S}_{i,\ell,h}, a^{L}_{i,\ell,h}\right). \tag{9}$$

Thus, different layer–head pairs can retain different token subsets, while the maximum across temporal views preserves tokens important to either recent or long-term attention. GrowPage releases exactly one page KV states at each compression event. Let P_t denote the current number of allocated pages, b the page size, and $\mathcal{R}_t$ the protected recent tokens. The resulting historical retention budget is therefore determined by the target capacity $(P_t - 1)b$, rather than treated as a tunable hyperparameter. We denote this derived budget by K_t. To prevent compression from over-concentrating on a few historical regions, GrowPage combines block-local retention with global selection independently for each layer–head pair. Let $\mathcal{B}_j$ denote the candidate positions from historical block j and k_{loc} the minimum local retention quota. For each (ℓ, h),

$$\begin{aligned} S^{\ell,h}_{\text{loc}} &= \bigcup_{j} \text{TopK}_{k_{\text{loc}}}(\mathcal{B}_j; s_{\ell,h}), \\ S^{\ell,h} &= S^{\ell,h}_{\text{loc}} \cup \text{TopK}_{K_t - |S^{\ell,h}_{\text{loc}}|}\left(\mathcal{C}_t \setminus S^{\ell,h}_{\text{loc}}; s_{\ell,h}\right), \end{aligned} \tag{10}$$

where $s_{\ell,h} = \{s_{i,\ell,h}\}_i$. Thus, k_{loc} controls only the balance between block-local coverage and global importance, while the total number of retained historical tokens is fixed by the requirement to create one page worth of free slots. The effect of k_{loc} is studied in Appendix D.2. The retained KV states are then compacted within the existing physical-page allocation.

Grow by One Page. When recent demand becomes more diffuse, GrowPage acquires one additional physical KV page instead of further compressing the historical working set. The page is appended to the request's block table, and decoding continues until the next capacity boundary.

4 System Integration

Section 3 describes how GrowPage estimates the evolving KV demand of each request and converts it into page-level capacity decisions. We now describe how these decisions are integrated into a PagedAttention-based serving runtime with low additional state overhead and minimal disruption to batched scheduling.

4.1 Lightweight Per-Request State Management

GrowPage introduces only a small amount of auxiliary state for online demand estimation. For each active request, the runtime maintains the short- and long-timescale query summaries described in Section 3.2. These states are allocated together with the request metadata, updated in place during decoding, and recycled when the request completes. Unlike the KV cache itself, their size is independent of sequence length, avoiding additional memory growth as reasoning proceeds.

To avoid frequent GPU-side memory allocation during continuous batching, GrowPage pre-allocates the storage required by these summaries for active request slots and reuses it across requests. We slightly modify the attention kernel to update the short- and long-timescale summaries from the normalized pre-RoPE queries during the regular decoding path. The more expensive attention-based demand estimation is invoked only at capacity boundaries, so most decoding steps require only lightweight in-place state updates.

4.2 Scheduler-Aware Page Allocation and Fallback

GrowPage maps the capacity decisions in Section 3.3 directly onto PagedAttention's physical-page abstraction. For *Compress & Hold*, the request keeps its existing physical-page allocation while selected historical KV states are compacted to release slots for subsequent tokens. For *Grow by One Page*, the scheduler obtains an additional page from the global free-page pool and appends its physical page identifier to the request's block table. The request can then continue decoding without changing the underlying attention or block-table interface.

Page growth is coordinated with global memory availability. A growth request is satisfied when a free physical page is available and the per-request capacity limit has not been reached. Otherwise, GrowPage falls back to compression within its current allocation. If compression requires additional target pages that are unavailable, the scheduler may preempt eligible uncompressed requests to obtain the required temporary storage. If allocation remains infeasible, the current request is deferred to a later scheduling round without changing its capacity. This keeps per-request adaptation subordinate to the global memory manager. Appendix E.2 provides the detailed preemption protocol.

Importantly, GrowPage preserves the page/block abstraction of PagedAttention rather than introducing a separate memory layout. It therefore remains compatible with serving mechanisms such as continuous batching, prefix caching, and CUDA Graph execution. More details see Appendix E. GrowPage additionally inherits the asynchronous compression pipeline of Zipage (Liao et al., 2026), allowing KV compaction to overlap with serving execution and reducing compression-induced stalls on the critical decoding path. Together, these design choices allow online KV-capacity adaptation to be incorporated into PagedAttention-based serving without sacrificing the system optimizations of the underlying engine.

5 Experiments

5.1 Experimental Setup

Models and benchmarks. We evaluate GrowPage on two representative reasoning models: DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025) and Qwen3-8B (Yang et al., 2025). Following

Table 1: Main comparison on reasoning benchmarks. We report pass@1 accuracy (%) and decoding throughput (tokens/s). The results are evaluated on DeepSeek-R1-Distill-Llama-8B and Qwen3-8B.

	Pass@1 Accuracy (%) ↑						TPS (tokens/s) ↑					
Method	GSM8K	MATH500	AMC23	AIME24	LiveCode	Avg.	GSM8K	MATH500	AMC23	AIME24	LiveCode	Avg.
DeepSeek-R1-Distill-Llama-8B												
FullKV (vLLM)	88.2	86.6	81.3	43.5	46.5	69.2	2705	1515	1235	766	1142	1472
FullKV (nano-vLLM)	88.0	86.2	80.9	43.2	46.1	68.9	2576	1442	1182	722	1086	1401
MorphKV (ICML'25)	85.5	75.8	70.4	35.8	33.9	60.3	507	235	189	119	178	245
R-KV (NeurIPS'25)	87.1	84.7	77.8	39.9	42.2	66.3	450	218	177	102	162	221
G-KV (arXiv'25)	87.7	85.1	78.3	40.5	42.8	66.9	472	228	185	111	171	233
Zipage (ACL'26)	88.0	85.0	79.0	41.4	44.6	67.6	3018	2174	2060	1158	1271	1936
GrowPage (Ours)	88.0	85.8	79.4	44.7	45.4	**68.7**	**3833**	**2599**	**2398**	**1465**	**1794**	**2417**
Qwen3-8B												
FullKV (vLLM)	95.7	96.1	92.5	75.5	83.9	88.7	2658	1382	1064	607	1076	1357
FullKV (nano-vLLM)	95.6	95.8	92.1	75.0	83.5	88.4	2486	1303	995	568	1015	1273
MorphKV (ICML'25)	94.2	82.5	71.8	57.5	64.8	74.2	463	425	321	168	330	341
R-KV (NeurIPS'25)	95.2	91.6	84.7	69.4	78.0	83.8	448	396	307	151	305	321
G-KV (arXiv'25)	95.1	92.5	84.2	68.9	78.6	83.9	424	408	309	154	317	322
Zipage (ACL'26)	95.7	93.7	89.5	72.7	80.7	86.5	3107	2076	1805	1007	1271	1853
GrowPage (Ours)	95.9	94.9	91.4	73.8	81.5	**87.5**	**3280**	**2553**	**2261**	**1190**	**1589**	**2174**

previous KV-cache compression studies, we consider both mathematical reasoning and code generation tasks, including GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), AMC23, AIME24, and LiveCodeBench (Jain et al., 2025). These benchmarks cover diverse reasoning workloads with varying output lengths and KV-cache demands.

Evaluation metrics and implementation. All experiments are conducted on a single NVIDIA A100 GPU with 80 GB memory. Following Zipage, we implement GrowPage on nano-vLLM with PagedAttention-based KV management and integrate the proposed capacity control mechanism into the decoding pipeline. Unless otherwise specified, we follow the same system configuration as Zipage, fixing the KV block size b to 256 tokens and the protected recent-window size $\mathcal{R}_t$ to 16 tokens. We fix the GPU memory utilization ratio to 0.9 for all vLLM/nano-vLLM-based experiments to ensure a consistent memory budget. We measure serving efficiency using decoding throughput (TPS), defined as the total number of output tokens generated across all requests divided by the wall-clock time from the start of decoding until all requests complete. Additional experimental details, including implementation settings and dataset configurations, are provided in Appendix F.1.

5.2 Main Results

Benchmark comparison. Table 1 compares GrowPage with full KV inference and representative KV-cache compression methods. For fixed-budget baselines, we use the dataset-specific KV budgets reported in Table 5. To keep the main comparison concise, Table 1 reports only Pass@1 accuracy and decoding throughput (TPS), while additional serving metrics and detailed analysis are provided in Appendix G. All serving-system baselines use the same GPU memory budget, while MorphKV, R-KV, and G-KV use the largest executable batch size under the same memory constraint to maximize throughput. GrowPage consistently achieves a better reasoning performance–throughput trade-off across both models. Compared with full KV inference, GrowPage improves the average TPS by 64.3% on DeepSeek-R1-Distill-Llama-8B and 60.2% on Qwen3-8B while maintaining comparable pass@1 accuracy. Although MorphKV, R-KV, and G-KV substantially reduce KV memory, their lack of serving-system co-design limits the resulting throughput gains even under maximized batching. Compared with Zipage, which preserves PagedAttention compatibility, GrowPage further improves throughput by adapting KV capacity to request-specific demand rather than relying on a fixed budget.

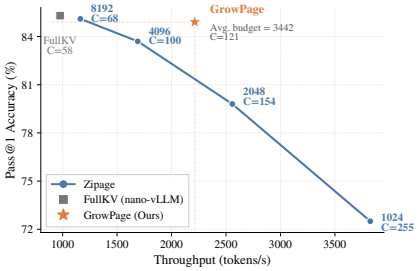


Figure 5: Accuracy–throughput trade-off on the mixed GSM8K, AMC23, and AIME24 workload with Qwen3-8B.

Pareto Analysis under Mixed Workloads. To evaluate on-demand KV budgeting under heterogeneous request demands, we construct a mixed workload combining GSM8K, AMC23, and AIME24, and compare GrowPage with FullKV and Zipage under different KV budgets. Details

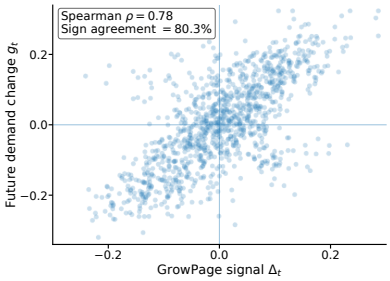


Figure 6: Correlation between the GrowPage signal Δ_t and future attention-demand change on Qwen3-8B with AMC23.

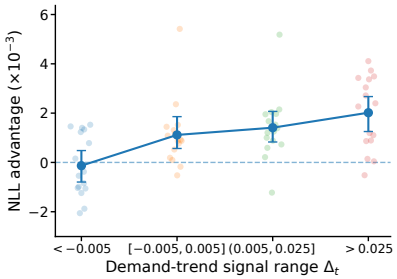


Figure 7: Counterfactual validation of GrowPage adaptive capacity decisions on Qwen3-8B with AMC23.

are provided in Appendix F.2. As shown in Figure 5, C denotes the average number of concurrent requests resident in GPU memory during serving. Increasing the fixed budget of Zipage improves accuracy but reduces C, resulting in lower throughput. GrowPage achieves 84.9% pass@1 accuracy at 2213 tokens/s with an average KV budget of 3442 and $C = 121$. It outperforms Zipage-4096 in accuracy (84.9% vs. 83.7%), throughput (2213 vs. 1690 tokens/s), and resident concurrency (121 vs. 100), while approaching the accuracy of Zipage-8192 (85.1%) with nearly $1.9\times$ higher throughput and $1.8\times$ more resident requests. These results show that adaptive KV budgeting better translates limited GPU memory into serving concurrency and throughput under heterogeneous workloads.

5.3 Analysis and Discussion

Future Demand Evolution Prediction. We first evaluate whether Δ_t reflects the actual change of future KV demand. Specifically, we define the future demand change as the variation between the current attention working set and the actual attention working set of tokens generated within the next physical KV page. A positive value indicates that future tokens require a broader historical working set, while a negative value indicates increasing attention concentration. Figure 6 shows the correlation between Δ_t and future demand change on Qwen3-8B with AMC23. The two variables exhibit a strong monotonic relationship, achieving a Spearman correlation coefficient of $\rho = 0.78$ and a sign agreement of 80.3%. This demonstrates that the dual-timescale query summaries effectively capture the temporal evolution of attention demand.

Capacity Expansion Benefit Analysis. We further investigate whether Δ_t can identify when allocating additional KV capacity is beneficial. At each capacity boundary, we construct two counterfactual KV states corresponding to *Compress & Hold* and *Grow by One Page*. To avoid the influence of different generation trajectories, we first obtain a reference decoding trace using full KV inference and replay the subsequent tokens under the two counterfactual KV states. Both branches share the same model parameters and future token sequence from the reference trajectory, differing only in the available KV capacity. We evaluate the prediction quality of the replayed tokens using teacher-forced negative log-likelihood (NLL). Let L_{compress} and L_{grow} denote the average NLL under *Compress & Hold* and *Grow by One Page*, respectively. We define the growth advantage as $A_t = L_{\text{compress}} - L_{\text{grow}}$, where a larger A_t indicates a larger reduction in prediction loss from allocating additional KV capacity. Figure 7 groups samples according to different Δ_t ranges and reports the corresponding growth advantage. The growth advantage consistently increases as Δ_t becomes larger, demonstrating that the proposed demand-trend signal not only captures attention-demand variation but also provides a reliable indicator for capacity expansion decisions.

Table 2: Ablation of dual-timescale demand estimation on Qwen3-8B.

β_S	β_L	Metric	AMC23		AIME24	
			Pass@1	TPS	Pass@1	TPS
0	0.999	Top-p	89.9	2394	71.2	1264
0	0.99	Top-p	89.1	**2478**	69.9	**1317**
0.9	0.99	Top-p	90.3	2362	71.7	1256
0.99	0.999	Top-p	90.9	2308	72.9	1218
0.9	0.999	Entropy	90.5	2179	72.4	1136
0.9	**0.999**	**Top-p**	**91.4**	2261	**73.8**	1190

Dual-Timescale Signal Ablation. We further ablate the design of the dual-timescale demand signal on Qwen3-8B, with results reported in Table 2. Here, $\beta_S = 0$ reduces the short-term summary to the current query. Across both AMC23 and AIME24, using the current query directly yields higher throughput but noticeably lower reasoning accuracy, while introducing moderate short-term smoothing improves the accuracy–throughput trade-off. A longer-term reference $\beta_L = 0.999$ also

consistently improves accuracy over $\beta_L = 0.99$ under the same short-term setting. However, overly smoothing the short-term summary $\beta_S = 0.99$ weakens its responsiveness and performs worse than the default $(\beta_S, \beta_L) = (0.9, 0.999)$, which achieves the highest pass@1 accuracy of 91.4% on AMC23 and 73.8% on AIME24. Replacing the Top-p working-set metric with attention entropy further degrades both accuracy and throughput under the same EMA configuration. These results support the use of well-separated temporal summaries together with the Top-p working set for robust online demand estimation.

Runtime Overhead Analysis. GrowPage invokes demand estimation and capacity control only at capacity boundaries rather than at every decoding step. Figure 8 shows that Online Demand, Attention Score, Window Mask, Top-k Selection, and KV Compaction incur only 5.84, 8.54, 0.70, 8.72, and 19.05 ms per event, respectively, accounting for less than 0.32% of the corresponding decoding time. KV Compaction dominates the additional overhead, while the overall runtime cost remains negligible relative to long-output decoding.

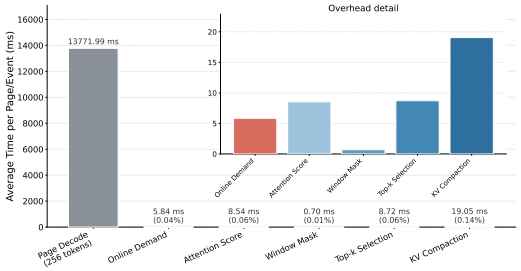


Figure 8: Runtime overhead analysis of GrowPage on Qwen3-8B with AMC23.

6 Related Work

KV-cache compression for LLM inference. KV-cache compression has become an important direction for reducing the memory footprint of long-context and reasoning inference. Early approaches mainly focus on retaining important tokens under a limited cache budget, such as StreamingLLM (Xiao et al., 2024), SnapKV (Li et al., 2024), and PyramidKV (Cai et al., 2024). Recent studies extend KV eviction to the decoding stage to control cache growth during long-form generation. Representative methods include H2O (Zhang et al., 2023), Quest (Tang et al., 2024), R-KV (Cai et al., 2026), G-KV (Liao et al., 2025), and ThinkKV (Ramachandran et al., 2026). These methods mainly improve token retention or eviction policies under a predefined KV budget. However, the allocated capacity itself is typically fixed during decoding, which limits their ability to adapt to heterogeneous and evolving KV demands across reasoning requests.

System-aware KV optimization. Beyond token selection algorithms, recent works explore integrating KV management with practical LLM serving systems. PagedAttention (Kwon et al., 2023) introduces a page-based memory abstraction that enables efficient KV management under continuous batching. Recent system-level KV management approaches further investigate efficient page organization and KV scheduling for long-sequence inference (Mao et al.). Zipage (Liao et al., 2026) integrates token-level KV eviction with PagedAttention through Compressed PagedAttention, preserving important serving features such as prefix caching. Nevertheless, existing system-compatible approaches still assume a predefined per-request KV capacity. In contrast, GrowPage treats KV capacity itself as a runtime resource and dynamically maps demand estimation into page-level allocation decisions.

7 Conclusion

Large reasoning models introduce substantial KV-cache pressure due to their long chain-of-thought generation, making efficient memory management critical for LLM reasoning serving. Existing KV compression methods typically rely on fixed per-request capacity budgets, which cannot accommodate the heterogeneous and evolving KV demands of reasoning requests. In this work, we present GrowPage, an on-demand KV budgeting framework that treats KV capacity as a runtime resource. GrowPage uses lightweight dual-timescale query summaries to estimate demand evolution and dynamically chooses between KV compaction within the current allocation and incremental physical-page expansion. By integrating this capability with PagedAttention-based serving, GrowPage preserves key system optimizations including continuous batching and prefix caching while achieving improved reasoning performance–throughput trade-offs across diverse benchmarks.

AI USE STATEMENT

Generative AI tools were used to assist with literature retrieval and analysis, English translation and editing, and drafting portions of the manuscript. Generative AI tools also assisted with mathematical derivations and proof writing. The authors take full responsibility for the final manuscript, including all scientific claims and reported results.

REPRODUCIBILITY STATEMENT

The method and system implementation are described in Section 3, Section 4 and Appendix E. The assumptions and proof for the theoretical analysis are provided in Section 3.1 and Appendix C. Model and hardware configurations, datasets, and evaluation protocols are reported in Section 5.1 and Appendix F.1. Additional hyperparameter and capacity-control analyses are provided in Appendix D and Appendix H. We will publicly release the source code for GrowPage upon acceptance.

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.

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.

Zefan Cai, Wen Xiao, Hanshi Sun, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, et al. R-kv: Redundancy-aware kv cache compression for reasoning models. *Advances in neural information processing systems*, 38:60980–61005, 2026.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. Nacl: A general and effective kv cache eviction framework for llm at inference time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7913–7926, 2024.

Pengyu Cheng, Jiacheng Wang, Tianle Chen, Bei Liu, Xiaofeng Hou, and Jiacheng Liu. Desirekv: Decoupling sensitivity and importance for reasoning-aware kv cache compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 20518–20526, 2026.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2026.

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*, volume 2024, pp. 22975–22988, 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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shen Han, Yuyang Wu, Junpu Yu, and Olexandr Isayev. Kara: Efficient reasoning llm serving via sliding-window kv cache compression. *arXiv preprint arXiv:2607.01237*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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.

Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024.

Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. Raas: Reasoning-aware attention sparsity for efficient llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2577–2590, 2025.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Jushi Kai, Zhuiri Xiao, Alexandra Birch, and Zhouhan Lin. Information-aware kv cache compression for long reasoning. *arXiv preprint arXiv:2606.26875*, 2026.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

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.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 155–172, 2024.

Mengjie Li, Yuan Feng, Xike Xie, and William J Song. Real: Retrieval-reasoning and logic-constructed attention behaviors for long-context kv cache compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39035–39052, 2026.

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.

Mengqi Liao, Lu Wang, Chaoyun Zhang, Zekai Shen, Xiaowei Mao, Si Qin, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Huaiyu Wan. G-kv: Decoding-time kv cache eviction with global attention. *arXiv preprint arXiv:2512.00504*, 2025.

Mengqi Liao, Lu Wang, Chaoyun Zhang, Bo Qiao, Si Qin, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Huaiyu Wan. Zipage: Maintain high request concurrency for llm reasoning through compressed pagedattention. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 7716–7737, 2026.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024a.

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.

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.

Yuzhen Mao, Qitong Wang, Martin Ester, and Ke Li. Efficient and accurate kv-cache management for long-sequence llms. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

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.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Transactions on Storage*, 22(4):1–38, 2026.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 110072–110106, 2026.

Yi Su, Zhenxu Tian, Dan Qiao, Yuechi Zhou, Juntao Li, and Min Zhang. Longflow: Efficient kv cache compression for reasoning models. *arXiv preprint arXiv:2603.11504*, 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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

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. In *International Conference on Learning Representations*, volume 2025, pp. 37228–37253, 2025.

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.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025.

Haoyue Zhang, Hualei Zhang, Xiaosong Ma, Jie Zhang, and Song Guo. Lazyeviction: Lagged kv eviction with attention pattern observation for efficient long reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36335–36352, 2026.

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.

Junqi Zhao, Zhijin Fang, Shu Li, Shaohui Yang, and Shichao He. Buzz: Beehive-structured sparse kv cache with segmented heavy hitters for efficient llm inference. *ACM Transactions on Intelligent Systems and Technology*, 17(4):1–22, 2026.

APPENDIX

A Overview of Mathematical Notation **15**

B Extended Related Works **15**

C Proof of Theorem 3.1 **16**

D Hyperparameter Sensitivity **17**

D.1 Effect of Demand Threshold 17

D.2 Effect of Local Retention Quota 17

E System Implementation Details **17**

E.1 Dual-Timescale Query Management and Online Demand Estimation 18

E.2 Prefix-Caching Support 18

E.3 Asynchronous Decoding and Compression 19

E.4 Preemption and Request Recovery 19

F Experimental Details **20**

F.1 Datasets and Evaluation Protocol 20

F.2 Mixed-Workload Construction 20

G Extended Analysis of Main Results **20**

G.1 Additional Mixed-Workload Results 20

G.2 Detailed Serving Metrics 21

H Additional Capacity-Control Analysis **22**

H.1 Alternative Capacity-Control Policies 22

H.2 System-Level Event Analysis 23

I Request-Level Capacity Alignment **24**

Table 3: Summary of key notation used in the paper.

Symbol	Description
$B_i^\star$	Minimum sufficient KV budget of request i
$\mathbf{a}_t$	Attention distribution over historical KV states at decoding step t
$D_p(\mathbf{a}_t)$	Minimum number of KV states required to cover attention mass p
S	Subset of retained historical KV states
$m_t(S)$	Attention mass retained by subset S
$\mathbf{o}_t, \widehat{\mathbf{o}}_t(S)$	Full-cache and subset-renormalized attention outputs
$C_\delta^{\mathrm{wc}}(\mathbf{a}_t)$	Minimum worst-case KV capacity under approximation tolerance δ
$\mathbf{q}_t$	Current normalized pre-RoPE query
$\bar{\mathbf{q}}_t^S, \bar{\mathbf{q}}_t^L$	Short- and long-timescale EMA query summaries
β_S, β_L	EMA decay factors for the two temporal query summaries
$\mathcal{C}_t$	Historical candidate KV set for online demand estimation
$K_{t,\ell,h}^x$	Top-p working-set size for temporal view $x \in \{S, L\}$
r_t^S, r_t^L	Request-level short- and long-timescale working-set ratios
Δ_t	Demand-trend signal, defined as $r_t^S - r_t^L$
τ	Threshold controlling compression versus capacity growth
$s_{i,\ell,h}$	KV importance score for token i at layer ℓ and head h
P_t	Number of physical KV pages allocated to the request
b	Number of KV-token slots in one physical page
$\mathcal{R}_t$	Protected recent-token set excluded from compression
K_t	Historical KV retention budget at a compression event
k_{loc}	Block-local KV retention quota

A Overview of Mathematical Notation

For clarity, Table 3 summarizes the key notation used in the theoretical analysis and the GrowPage capacity-control framework.

B Extended Related Works

Fine-Grained KV Compression. Beyond the representative KV-eviction methods discussed in the main text, a broad body of work explores finer-grained cache adaptation and alternative compression dimensions. Scissorhands (Liu et al., 2023) exploits the persistence of token importance to probabilistically retain historically influential KV states, while Keyformer (Adnan et al., 2024) identifies a compact set of key tokens that dominate attention during generation. Rather than applying a uniform policy to all attention heads, FastGen (Ge et al., 2024) profiles their attention structures and assigns different cache strategies accordingly, and DuoAttention (Xiao et al., 2025) separates retrieval heads requiring long-context access from streaming heads that can operate with compact caches. NACL (Chen et al., 2024) combines proxy-token-based importance with randomized eviction, whereas BUZZ (Zhao et al., 2026) preserves recent context together with segmented historical heavy hitters. A complementary direction reduces the representation cost of retained KV states. KIVI (Liu et al., 2024c) introduces asymmetric low-bit quantization tailored to the distinct distributions of keys and values, and KVQuant (Hooper et al., 2024) combines pre-RoPE key quantization, non-uniform datatypes, and explicit outlier handling. MiKV (Yang et al., 2024) preserves important KV states at higher precision while representing less important states at lower precision, and GEAR (Kang et al., 2024) combines quantization with low-rank and sparse error correction. Cross-layer redundancy has also been exploited by MiniCache (Liu et al., 2024a), which merges similar KV states across neighboring layers. These approaches adapt *which* states are represented and *how* they are represented, but do not directly address online request-level decisions about how much total KV capacity should be allocated as generation evolves.

KV Compression for Long Reasoning. Recent work has increasingly specialized KV-cache compression for long-output reasoning, where token importance can change substantially over an extended generation trajectory. LazyEviction (Zhang et al., 2026) identifies *token importance recurrence*, observing that previously unimportant tokens may regain high attention after many decoding steps, and therefore delays irreversible eviction through an observation window. DesireKV (Cheng et al., 2026) jointly considers attention-derived importance and quantization sensitivity, selectively retaining, quantizing, or evicting reasoning tokens according to their different roles. LongFlow (Su et al., 2026) derives token-importance estimates directly from intermediate attention computations and co-designs them with fused kernels to reduce the overhead of continuous importance evaluation during long reasoning. InfoKV (Kai et al., 2026) further argues that attention alone may overlook tokens with long-range future influence and supplements attention scores with information-theoretic signals related to predictive uncertainty. Kara (Han et al., 2026) restricts compression to a sliding window of recently generated reasoning context and develops a serving implementation compatible with paged KV management. REAL (Li et al., 2026) instead analyzes heterogeneous attention-head behaviors in both successful and failed retrieval-reasoning cases to construct more robust eviction policies. Together, these studies reinforce that reasoning-time KV importance is highly dynamic. Nevertheless, their adaptation primarily concerns importance estimation, eviction timing, token grouping, or representation precision; GrowPage operates on an orthogonal dimension by adapting the *total per-request capacity* according to the evolving demand of the request.

System-Level KV Management. Another related direction optimizes where and how KV states are physically stored and transferred in serving systems. vAttention (Prabhu et al., 2025) uses CUDA virtual memory to support dynamic physical-memory allocation while retaining a contiguous virtual KV layout, illustrating that the physical storage backing an active sequence can be managed independently from its logical address space. For memory hierarchies extending beyond GPU memory, InfiniGen (Lee et al., 2024) predicts important KV entries and selectively fetches them from CPU memory, thereby reducing the transfer overhead of conventional KV offloading. CacheGen (Liu et al., 2024b) compresses reusable KV caches into compact bitstreams for efficient context loading and adapts compression to network conditions. At cluster scale, MemServe (Hu et al., 2024) introduces an elastic distributed memory pool for context caching and disaggregated inference, while Mooncake (Qin et al., 2026) builds a KV-cache-centric serving architecture that exploits heterogeneous CPU, DRAM, SSD, and network resources together with a global scheduler. These systems mainly improve KV placement, movement, reuse, or physical memory provisioning. GrowPage addresses a different but complementary problem: before allocating additional GPU-resident KV storage to an active reasoning request, it estimates whether the request's evolving attention demand justifies expanding its logical cache capacity. This demand-aware capacity control allows the page-based memory manager to respond not only to sequence growth, but also to heterogeneous and time-varying information requirements.

C Proof of Theorem 3.1

For a retained subset S, let

$$m(S) = \sum_{i \in S} a_{t,i}.$$

The difference between the full and renormalized attention outputs is

$$\mathbf{o}_t - \widehat{\mathbf{o}}_t(S) = \sum_{i \notin S} a_{t,i}\mathbf{v}_i - \frac{1 - m(S)}{m(S)} \sum_{i \in S} a_{t,i}\mathbf{v}_i.$$

Using $\|\mathbf{v}_i\|_2 \le V_{\max}$ and the triangle inequality,

$$\begin{aligned} \|\mathbf{o}_t - \widehat{\mathbf{o}}_t(S)\|_2 &\le \sum_{i \notin S} a_{t,i} V_{\max} + \frac{1 - m(S)}{m(S)} \sum_{i \in S} a_{t,i} V_{\max} \\ &= 2V_{\max}(1 - m(S)). \end{aligned}$$

This bound is tight. Let $\mathbf{u}$ be any unit vector and choose $\mathbf{v}_i = -V_{\max}\mathbf{u}$ for $i \in S$ and $\mathbf{v}_i = V_{\max}\mathbf{u}$ for $i \notin S$. Then

$$\|\mathbf{o}_t - \widehat{\mathbf{o}}_t(S)\|_2 = 2V_{\max}(1 - m(S)).$$

Hence,

$$\sup_{\|\mathbf{v}_i\|_2 \le V_{\max}} \|\mathbf{o}_t - \widehat{\mathbf{o}}_t(S)\|_2 = 2V_{\max}(1 - m(S)).$$

Therefore, guaranteeing error at most δ for all admissible value vectors is equivalent to

$$m(S) \ge 1 - \frac{\delta}{2V_{\max}}.$$

Among all subsets of cardinality k, the maximum retained attention mass is obtained by selecting the k largest attention weights. Thus, the minimum cardinality satisfying the above condition is

$$C_\delta^{\mathrm{wc}}(\mathbf{a}_t) = D_{1-\delta/(2V_{\max})}(\mathbf{a}_t),$$

which proves the theorem.

D Hyperparameter Sensitivity

D.1 Effect of Demand Threshold

We further study the sensitivity of GrowPage to the demand threshold τ, which controls the trade-off between KV compression and capacity expansion. As shown by the counterfactual analysis in Figure 7, the benefit of allocating additional KV capacity changes noticeably as Δ_t crosses the region around zero. We therefore focus on the fine-grained interval around $[-0.005, 0.005]$ and vary τ from -0.003 to 0.003 with a step size of 0.001. All experiments are conducted on Qwen3-8B using AMC23 and AIME24. Figure 9 shows that τ provides a direct mechanism for controlling the accuracy–throughput operating point of GrowPage. A smaller threshold makes capacity expansion more permissive, causing more requests to acquire additional KV pages. This preserves a larger historical working set and generally improves reasoning accuracy, but increases per-request KV consumption and consequently reduces serving throughput. Conversely, a larger τ makes GrowPage more conservative about expansion, resulting in more frequent *Compress & Hold* decisions and higher throughput at the cost of more aggressive information removal. Across both AMC23 and AIME24, $\tau = 0$ provides a stable operating point with a balanced trade-off between reasoning accuracy and serving throughput. We therefore adopt $\tau = 0$ as the default threshold throughout our experiments.

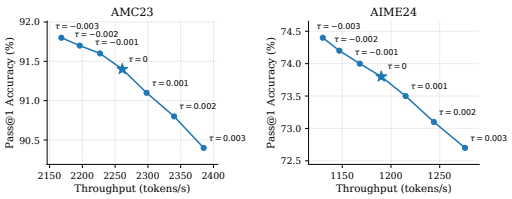


Figure 9: Accuracy–throughput trade-offs under different demand thresholds τ on Qwen3-8B with AMC23 and AIME24.

D.2 Effect of Local Retention Quota

As described in Section 3.3, k_{loc} specifies the minimum number of tokens retained from each historical block before the remaining budget is filled by global importance-based selection. We evaluate its sensitivity on Qwen3-8B using AMC23 and AIME24. We report only Pass@1 because different k_{loc} values yield nearly identical TPS; k_{loc} mainly affects which KV states are retained rather than the total retained capacity, and therefore primarily influences reasoning accuracy. As shown in Table 4, performance first improves as k_{loc} increases from 16 to 64, indicating that sufficient block-local coverage helps prevent globally important regions from dominating the retained KV states. However, further increasing k_{loc} to 128 slightly degrades accuracy, since an overly large local quota leaves less flexibility for global selection and forces more tokens to be preserved from relatively unimportant blocks. Overall, $k_{\mathrm{loc}} = 64$ achieves the best performance on both benchmarks, and we therefore use it as the default setting throughout our experiments.

Table 4: Effect of the local retention quota k_{loc} on Qwen3-8B.

k_{loc}	AMC23 Pass@1	AIME24 Pass@1
16	90.3	72.4
32	90.9	73.1
64	**91.4**	**73.8**
128	90.6	72.8

E System Implementation Details

This section provides additional implementation details for the system integration described in Section 4. We first describe the runtime management and computation of the dual-timescale query

states. Our support for prefix caching and asynchronous KV compression largely follows the system design of Zipage (Liao et al., 2026), with the compression target adapted to GrowPage's dynamically determined page capacity.

E.1 Dual-Timescale Query Management and Online Demand Estimation

Pre-allocated query-state pool. To avoid request-level GPU allocation and deallocation, GrowPage pre-allocates a fixed query-state tensor when the inference engine is initialized. Its layout is

$$\mathcal{Q} \in \mathbb{R}^{L \times M \times 2 \times H_q \times d_h}, \tag{11}$$

where L is the number of attention layers, M is the maximum number of query slots, H_q is the number of query heads per GPU, and d_h is the head dimension. The third dimension stores the short- and long-timescale EMA queries. We set M to the serving engine's maximum number of concurrent sequences (`max_num_seqs`), which is 512 by default.

Query slots are bound to requests only when they enter the running queue. The scheduler maintains a pool of free slot identifiers and assigns one identifier to each newly admitted running request; requests waiting to be scheduled therefore consume no query-state slot. When a request finishes, its identifier is returned to the free pool without releasing the underlying GPU tensor. The same procedure is applied when a request is preempted; if it is scheduled again, it acquires a new slot and reinitializes its summaries. This slot-based design avoids frequent CUDA memory allocation and allows the pre-allocated storage to be continuously reused across requests. The resulting auxiliary memory is fixed with respect to sequence length and scales as $2MLH_qd_h$ elements.

Initialization and online update. The query summaries are updated only during decoding. For the first decoding step of a newly assigned slot, both temporal states are initialized directly from the current normalized pre-RoPE query, $\bar{\mathbf{q}}^S = \bar{\mathbf{q}}^L = \mathbf{q}_t$, thereby overwriting any stale values left by the previous owner of the slot without explicitly clearing the tensor. Subsequent decoding steps update the two states in place according to Eq. 6, using $\beta_S = 0.9$ and $\beta_L = 0.999$. No historical query sequence is materialized: each request stores only the two running summaries.

Demand estimation at capacity boundaries. The regular decoding path only updates the two EMA states. The remaining demand-estimation operations are executed when a request reaches a capacity boundary. Because EMA averaging changes the magnitude of the stored queries, we first calibrate each temporal summary to the RMS scale of the current normalized query. For $x \in \{S, L\}$, we compute

$$\gamma_t^x = \frac{\mathrm{RMS}(\mathbf{q}_t)}{\mathrm{RMS}(\bar{\mathbf{q}}_t^x)}, \qquad \widetilde{\mathbf{q}}_t^x = \gamma_t^x \bar{\mathbf{q}}_t^x. \tag{12}$$

The calibrated queries are then transformed by RoPE using the *current* decoding position. This preserves the temporal information accumulated in the common pre-RoPE space while aligning both summaries with the positional frame used by the current attention computation.

We exclude the most recent physical KV block from the historical candidate set, corresponding to 256 tokens under our default block size, and independently compute the short- and long-timescale attention distributions over the remaining historical keys. The native GQA head mapping is preserved when matching query heads to KV heads. For each temporal view, the attention weights are sorted and accumulated until reaching the Top-p coverage threshold ($p = 0.99$), yielding the working-set ratios in Eq. 7. These ratios are then aggregated across layers and KV heads according to Eq. 8, producing the request-level signal Δ_t. Finally, Δ_t is compared with the demand threshold τ to select between *Compress & Hold* and *Grow by One Page*. Thus, attention scoring, sorting, and working-set estimation are kept off the critical path of ordinary decoding steps and are invoked only when a capacity decision is required.

E.2 Prefix-Caching Support

GrowPage inherits the prefix-preserving compression mechanism of Zipage (Liao et al., 2026). Prefix sharing is tracked at physical-block granularity using reference counts, and a block referenced by multiple requests is never overwritten in place during compression. Instead, compression writes the

retained KV states into a set of target blocks while leaving shared prefix blocks intact for the other requests. We adapt the target size to GrowPage's dynamic capacity. For a request currently occupying P_t physical pages, *Compress & Hold* keeps the physical allocation unchanged but compacts the retained KV states into at most $(P_t - 1)b$ occupied slots, leaving one page worth of free slots for subsequent decoding. If shared prefix blocks occupy part of the original allocation, sufficient new target blocks are allocated to replace those shared blocks, while reusable non-shared blocks are retained whenever possible. After compaction, the request releases its references to the original shared blocks and their reference counts are updated accordingly. *Grow by One Page* does not modify the shared prefix and simply appends the newly allocated page to the request's block table. As in Zipage, temporary target-block allocation may precede the release of shared blocks. When the free-page pool cannot satisfy this allocation, GrowPage applies the preemption and deferred execution procedure described in Appendix E.4.

E.3 Asynchronous Decoding and Compression

We also adopt the asynchronous decoding–compression pipeline of Zipage (Liao et al., 2026). At a given scheduling step, only a subset of requests reaching capacity boundaries require KV compaction. Executing compression synchronously would stall the remaining decode-ready requests and serialize a relatively small compression batch with the main decoding workload. Instead, requests selected for *Compress & Hold* enter the asynchronous compression path, while other ready requests continue decoding. Once compaction completes, the compressed requests rejoin a subsequent decoding batch. Requests taking *Grow by One Page* require only page allocation and block-table update and can resume decoding after the new page is installed. This preserves Zipage's overlap between decoding and KV compaction while allowing GrowPage's capacity decisions to remain request-specific.

E.4 Preemption and Request Recovery

Capacity decisions and temporary allocation. Page growth uses only pages available in the global free-page pool and does not trigger request preemption. When the per-request capacity limit has been reached or no free page is available, a growth decision falls back to compression at the current capacity. For both policy-selected compression and fallback compression, the scheduler first determines the additional target pages required for compaction. If these pages are available, compaction proceeds directly; otherwise, the scheduler applies the preemption rule below. Successful Compress & Hold preserves the current capacity, whereas a successful Shrink action in GrowPage-Shrink reduces it by one page. Compression entered through a growth fallback always preserves the current capacity.

Preemption eligibility and deferred execution. Eligible requests must be running, distinct from the request awaiting compression, and have never completed KV compression. Requests with a pending compression operation or capacity decision are excluded. Once a request completes its first compression, it remains ineligible for preemption for the rest of its execution. The scheduler scans the running queue from its tail and preempts eligible requests one at a time, rechecking the target-page requirement after each removal and stopping as soon as the requirement is satisfied. If sufficient space cannot be obtained from eligible requests, the current request performs neither compression nor decoding in that scheduling round. Its capacity remains unchanged, and the operation is retried in a subsequent round.

Request preservation and recovery. Preemption returns a request to the waiting queue and releases its KV-block references and query-state slot. Shared physical blocks become available only after all references to them are released. The request retains its complete token sequence, including the original prompt and all generated tokens, together with its identity, capacity configuration, sampling seed, generation-step progress, and token and timing metadata. Upon readmission, the retained sequence is processed through prefill to reconstruct the KV cache, reusing valid complete prefix blocks when prefix caching is enabled. Generation then resumes at the next token position without regenerating the retained output. The request acquires a query-state slot, and both EMA summaries are reinitialized from the first resumed decoding query, following Appendix E.1. Since

Table 5: Evaluation settings for individual reasoning benchmarks. The KV budget denotes the per-request capacity used by fixed-budget compression baselines.

Workload	Number of Questions	Sample Times	Max Output Length	KV Budget
AMC23	40	32	16384	4096
AIME24	30	32	32768	4096
MATH500	500	8	16384	4096
GSM8K	1319	4	16384	4096
LiveCodeBench v1	400	8	16384	8192

requests that have undergone compression are ineligible for preemption, this recovery procedure does not restore KV states previously discarded by compression.

F Experimental Details

F.1 Datasets and Evaluation Protocol

We evaluate GrowPage on five reasoning benchmarks covering mathematical reasoning and code generation. Table 5 summarizes the dataset size, number of independent generations per question, maximum output length, and the KV budget used by fixed-budget compression baselines. Unless otherwise specified, we use a sampling temperature of 0.6. For benchmarks with multiple generations per question, each generation is evaluated independently and the reported Pass@1 is averaged over all sampled outputs. The dataset-specific KV budgets follow the configurations used in our fixed-budget comparisons; GrowPage does not use these predefined budgets and instead adapts its KV capacity online.

The sampling multiplicity is chosen according to the size and difficulty of each benchmark. For the smaller competition-level mathematics datasets AMC23 and AIME24, we use 32 independent generations per question to reduce sampling variance. For the larger MATH500, GSM8K, and LiveCodeBench datasets, fewer repetitions provide sufficient coverage while keeping the total evaluation cost manageable. The larger KV budget for LiveCodeBench reflects its substantially longer code-generation trajectories.

F.2 Mixed-Workload Construction

For the heterogeneous-workload experiment in Figure 5, we combine AMC23, AIME24, and GSM8K into a single serving workload. Because these datasets differ substantially in the number of available questions, directly using each question once would cause GSM8K to dominate the workload. We therefore use different sampling multiplicities to obtain a more balanced contribution from the three datasets. The resulting workload contains 3559 requests, as summarized in Table 6. All requests use a maximum output length of 32768 tokens.

This construction prevents the much larger GSM8K dataset from overwhelmingly dominating the mixed workload and yields comparable numbers of requests from the three constituent benchmarks. Unlike the individual benchmark comparison in Table 5, we do not assign a single fixed KV budget to this workload, since the purpose of this experiment is to explicitly evaluate Zipage across multiple fixed budgets and compare its resulting accuracy–throughput Pareto frontier with GrowPage.

Table 6: Composition of the mixed reasoning workload.

Dataset	Number of Questions	Samples	Total Requests
AMC23	40	32	1280
AIME24	30	32	960
GSM8K	1319	1	1319

G Extended Analysis of Main Results

G.1 Additional Mixed-Workload Results

To examine whether the mixed-workload behavior in Figure 5 generalizes across model architectures, we repeat the same experiment on DeepSeek-R1-Distill-Llama-8B. We use the identical mixed workload described in Appendix F.2, and compare GrowPage with FullKV and Zipage under fixed KV budgets ranging from 1024 to 8192 tokens.

Table 7: Extended efficiency comparison on reasoning benchmarks. We report total inference time (hours) and time per output token (TPOT, ms/token); lower is better for both metrics.

	Total Inference Time (h) ↓						TPOT (ms/token) ↓					
Method	GSM8K	MATH500	AMC23	AIME24	LiveCode	Avg.	GSM8K	MATH500	AMC23	AIME24	LiveCode	Avg.
DeepSeek-R1-Distill-Llama-8B												
FullKV (vLLM)	0.61	3.30	2.00	5.27	8.10	3.86	117	522	994	783	579	599.0
FullKV (nano-vLLM)	0.64	3.44	2.06	5.25	8.51	3.98	123	546	1027	814	598	621.6
MorphKV (ICML'25)	3.95	25.99	16.09	44.49	66.13	31.33	136	157	156	241	248	187.6
R-KV (NeurIPS'25)	4.25	25.34	15.52	40.77	64.38	30.05	151	169	167	254	272	202.6
G-KV (arXiv'25)	4.10	24.59	14.78	37.33	60.81	28.32	145	163	162	247	260	195.4
Zipage (ACL'26)	0.55	2.63	1.23	3.48	7.75	3.13	81	128	127	240	225	160.2
GrowPage (Ours)	**0.49**	**2.18**	**1.19**	**2.93**	**5.75**	**2.51**	**78**	**110**	**112**	**207**	**155**	**132.4**
Qwen3-8B												
FullKV (vLLM)	1.23	4.09	2.66	6.58	6.35	4.18	139	381	826	938	496	556.0
FullKV (nano-vLLM)	1.32	4.33	2.82	7.01	6.73	4.44	148	404	872	1003	525	590.4
MorphKV (ICML'25)	8.29	17.08	10.76	35.64	26.69	19.69	170	165	169	250	232	197.2
R-KV (NeurIPS'25)	8.41	16.63	10.13	30.94	25.68	18.36	176	181	181	263	254	211.0
G-KV (arXiv'25)	8.63	16.01	10.23	31.25	24.33	18.09	181	174	176	256	246	206.6
Zipage (ACL'26)	1.06	3.11	1.63	4.19	5.39	3.08	79	141	138	249	177	156.8
GrowPage (Ours)	**1.04**	**2.60**	**1.42**	**3.77**	**4.67**	**2.70**	**75**	**106**	**122**	**207**	**145**	**131.0**

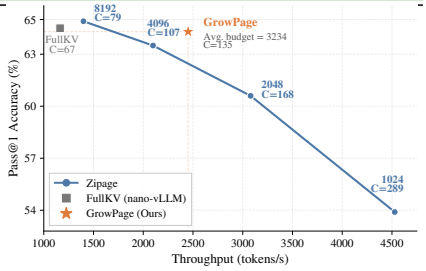


Figure 10: Accuracy–throughput trade-off on the mixed GSM8K, AMC23, and AIME24 workload with DeepSeek-R1-Distill-Llama-8B. C denotes the average number of requests resident in GPU memory.

As shown in Figure 10, DeepSeek-R1-Distill-Llama-8B exhibits a trend consistent with the Qwen3-8B results in Figure 5. Increasing the fixed KV budget of Zipage improves reasoning accuracy but progressively reduces resident concurrency and serving throughput. GrowPage instead reaches 64.3% Pass@1 at 2451 tokens/s with an average KV budget of 3234 tokens and $C = 135$. Compared with Zipage-4096, GrowPage improves accuracy from 63.5% to 64.3% while increasing throughput from 2098 to 2451 tokens/s and resident concurrency from 107 to 135. Compared with the more conservative Zipage-8192 setting, GrowPage remains within 0.6 percentage points in accuracy while achieving $1.75\times$ higher throughput and $1.71\times$ more resident requests. It also closely matches FullKV accuracy (64.3% vs. 64.5%) while providing $2.11\times$ higher throughput. These results further show that on-demand KV budgeting achieves a favorable accuracy–efficiency trade-off across different reasoning-model architectures rather than being specific to Qwen3-8B.

G.2 Detailed Serving Metrics

Table 7 provides additional efficiency metrics for the main comparison in Table 1. We report the total inference time for each benchmark and the time per output token (TPOT). The total time is converted from seconds to hours for readability. Since different benchmarks contain different numbers of sampled requests, total inference time should be compared across methods within the same benchmark rather than across datasets.

End-to-end efficiency. Table 7 further confirms that the throughput improvements of GrowPage translate into shorter workload completion time. GrowPage achieves the lowest total inference time across all five benchmarks on both models. Compared with Zipage, its average inference time decreases from 3.13 to 2.51 hours on DeepSeek-R1-Distill-Llama-8B and from 3.08 to 2.70 hours on Qwen3-8B. The improvement becomes particularly pronounced on longer and more demanding workloads; for example, on LiveCodeBench with DeepSeek-R1-Distill-Llama-8B, GrowPage reduces the total inference time from 7.75 to 5.75 hours. These results indicate that adapting per-request KV capacity reduces unnecessary memory residency and enables the serving engine to complete batched reasoning workloads more efficiently.

Per-token decoding efficiency. GrowPage also consistently achieves the lowest TPOT across all evaluated datasets. Averaged over the five benchmarks, GrowPage reduces TPOT from 160.2 to 132.4 ms/token relative to Zipage on DeepSeek-R1-Distill-Llama-8B, and from 156.8 to 131.0 ms/-token on Qwen3-8B. This suggests that on-demand capacity control not only increases aggregate concurrency but also reduces the attention cost associated with unnecessarily large KV working

Table 8: Comparison of alternative capacity-control policies on Qwen3-8B with AMC23. Total time denotes the end-to-end inference time of the workload. Lower is better for Avg. KV Budget, TPOT, and Total Time.

Method	Avg. KV Budget ↓	Pass@1 (%) ↑	TPS (tokens/s) ↑	TPOT (ms/token) ↓	Total Time (h) ↓
GROWPAGE-FIXED	4096	89.9	1874	136	1.59
GROWPAGE-SHRINK	**2099**	84.2	**2546**	**107**	1.44
GROWPAGE-RANDOM	3728	88.3	2008	137	1.57
GROWPAGE-INVERSE	6167	90.8	1340	175	2.13
Zipage (ACL'26)	4096	89.5	1805	138	1.63
GrowPage (Ours)	3164	**91.4**	2261	122	**1.42**

sets. Interestingly, MorphKV, R-KV, and G-KV achieve substantially lower TPOT than FullKV on several long-generation workloads, yet exhibit much lower aggregate TPS in Table 1. This discrepancy highlights that reducing per-token computation alone does not necessarily translate into higher serving throughput: without system-level integration with continuous batching and paged memory management, algorithmic KV compression may fail to efficiently exploit the freed memory under high concurrency. GrowPage combines both advantages, maintaining low per-token latency while translating KV savings into higher aggregate serving throughput.

H ADDITIONAL CAPACITY-CONTROL ANALYSIS

H.1 ALTERNATIVE CAPACITY-CONTROL POLICIES

We further investigate whether the performance gains of GrowPage arise specifically from its demand-guided capacity control, rather than merely from changing the KV capacity or the overall frequency of compression and growth. We compare the default GrowPage policy with four alternative capacity-control strategies on Qwen3-8B with AMC23.

Compared policies. GROWPAGE-FIXED disables capacity expansion and maintains a fixed KV budget of 4096 tokens; whenever the current capacity is exhausted, it applies *Compress & Hold*. GROWPAGE-RANDOM removes the dependence on Δ_t and randomly selects *Compress & Hold* with probability 0.7 and *Grow by One Page* otherwise. The probability is chosen to approximately match the empirical compression frequency of the default GrowPage policy, providing a frequency-matched control for evaluating whether the timing of capacity decisions matters. GROWPAGE-INVERSE reverses the default interpretation of the demand signal: it grows when $\Delta_t \leq 0$ and compresses when $\Delta_t > 0$. Finally, GROWPAGE-SHRINK introduces explicit bidirectional resizing with three actions:

$$\pi_{\text{shrink}}(\Delta_t) = \begin{cases} \textit{Shrink}, & \Delta_t < -0.005, \\ \textit{Compress \& Hold}, & -0.005 \leq \Delta_t \leq 0, \\ \textit{Grow by One Page}, & \Delta_t > 0. \end{cases} \tag{13}$$

Here, *Shrink* compacts two pages worth of historical KV states and returns one physical page to the global memory pool, while leaving one page worth of free slots for subsequent decoding. We impose a minimum KV capacity of 1024 tokens to prevent excessive contraction. The threshold -0.005 is motivated by the low-demand region identified in the counterfactual analysis of Figure 7.

Effect of demand-guided adaptation. Table 8 shows that dynamically adapting KV capacity is important for obtaining a favorable accuracy–efficiency trade-off. Compared with GROWPAGE-FIXED, the default policy reduces the average KV budget from 4096 to 3164 tokens while improving TPS from 1874 to 2261 and Pass@1 from 89.9% to 91.4%. This indicates that the benefit cannot be reproduced by simply maintaining a static capacity and repeatedly compressing within it.

More importantly, GROWPAGE-RANDOM produces an action frequency close to that of the default policy, but performs substantially worse. Despite using a similar fraction of compression and growth decisions, the random policy reaches only 88.3% Pass@1 at 2008 tokens/s, compared with 91.4% at 2261 tokens/s for GrowPage. It also maintains a larger average KV budget of 3728 tokens. This result shows that the gain does not arise merely from the marginal frequency of compression and growth; *when* each action is applied is critical.

Table 9: Runtime capacity-control events on Qwen3-8B with AMC23. Grow Ratio is computed over primary capacity decisions before fallback handling. Fallback denotes conversion of a growth decision to compression at the current capacity. Preemptions count eligible uncompressed requests removed to provide temporary storage for compression. For GROWPAGE-SHRINK, explicit Shrink events are reported separately from Compress & Hold.

Method	Grow	Grow Ratio (%)	Compress & Hold	Shrink	Fallback	Preemptions
GROWPAGE-FIXED	0	0.0	22285	–	0	19
GROWPAGE-SHRINK	17909	45.5	12540	8927	103	4668
GROWPAGE-RANDOM	9345	30.4	21399	–	944	1894
GROWPAGE-INVERSE	19881	66.7	9931	–	286	2997
GrowPage (Ours)	9500	30.9	21203	–	597	1361

Reversing the demand signal leads to the opposite behavior. GROWPAGE-INVERSE expands much more aggressively, increasing the average KV budget to 6167 tokens and reducing throughput to 1340 tokens/s, while still achieving lower accuracy than the default GrowPage policy. This result complements the counterfactual analysis in Figure 7 and confirms that the direction of Δ_t carries meaningful information for capacity allocation rather than serving only as a generic activity signal.

Why not aggressively shrink capacity? GROWPAGE-SHRINK achieves the smallest average KV budget and the highest raw throughput, but its Pass@1 drops sharply to 84.2%, 7.2 percentage points below the default GrowPage policy. The result exposes the cost of aggressively reclaiming physical pages from active reasoning requests: once additional historical KV states are irreversibly removed, later reasoning stages may again require information that is no longer available. GrowPage therefore adopts the more conservative *Compress & Hold* operation in the main design, retaining the current physical allocation while creating room for subsequent tokens. This yields a substantially better balance between memory efficiency, serving throughput, and reasoning accuracy.

H.2 SYSTEM-LEVEL EVENT ANALYSIS

To better understand how different capacity-control policies affect the serving runtime, we record page-growth, compression, fallback, and request-preemption events. A fallback occurs when a growth decision cannot be executed because the current capacity has reached its limit or the free-page pool is empty, and is therefore converted to compression at the current capacity. A preemption occurs when an eligible uncompressed request is removed from the running queue to satisfy the temporary allocation required by compression, following Appendix E.4. This can occur for either policy-selected compression or fallback compression. Consequently, preemption does not require a preceding fallback, and a single compression operation may require multiple request preemptions. For GROWPAGE-SHRINK, explicit Shrink events are included in the runtime's total compression count but are reported separately from Compress & Hold in Table 9.

Decision frequency versus decision timing. The runtime statistics provide a particularly controlled comparison between GrowPage and GROWPAGE-RANDOM. The two policies have almost identical primary action distributions: GrowPage selects Grow by One Page at 30.9% of capacity boundaries, compared with 30.4% for GROWPAGE-RANDOM, before fallback handling. Their total numbers of primary capacity decisions are also nearly identical. Nevertheless, GrowPage achieves substantially higher accuracy and throughput in Table 8. This frequency-matched comparison further demonstrates that the effectiveness of GrowPage comes from using Δ_t to determine *where* expansion is needed, rather than merely choosing an appropriate global compression-to-growth ratio. The demand-guided policy is also accompanied by fewer fallback and preemption events than its random counterpart.

Cost of dynamic capacity adaptation. Dynamic capacity adaptation changes the occupancy of the shared KV pool and the temporary storage available for compression. Under our scheduling rule, preemption is used to obtain target pages required by compression; page growth itself does not trigger preemption. Preemption can also occur under GROWPAGE-FIXED, since fixed-capacity compaction may require additional target pages to preserve shared prefixes. The events in Table 9 therefore characterize the interaction between capacity control and compression allocation, complementing the accuracy, throughput, and average KV budget measurements in Table 8.

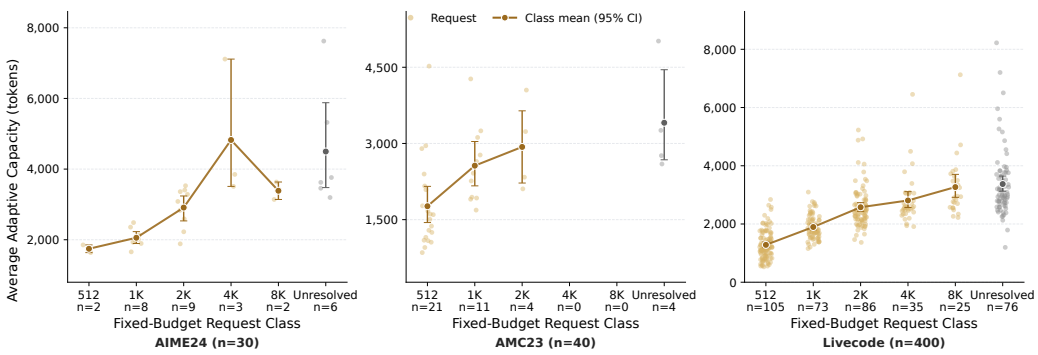


Figure 11: Alignment between minimum sufficient fixed-budget classes and GrowPage's adaptive KV capacity. Requests are grouped by the minimum sufficient budget identified in Figure 2, and each point shows the average capacity allocated by GrowPage to the same request under the same random seed. Markers and error bars denote the class mean and 95% confidence interval.

Capacity over-expansion and bidirectional resizing. The event distribution of GROWPAGE-INVERSE further illustrates the importance of the direction of the demand signal. Reversing the policy raises the growth ratio from 30.9% to 66.7%, leading to substantially larger average KV residency and lower serving throughput. Although this policy retains more KV information, the additional capacity does not translate into higher reasoning accuracy, indicating that indiscriminate expansion wastes shared GPU memory.

GROWPAGE-SHRINK exhibits a different failure mode. It triggers 8927 explicit shrink operations while also performing 17909 subsequent growth operations, resulting in substantially more bidirectional capacity transitions. This aggressive resizing is accompanied by 4668 request preemptions, considerably more than the 1361 observed with the default GrowPage policy. Together with the significant accuracy degradation in Table 8, these results suggest that aggressively reclaiming pages from live reasoning requests can introduce unnecessary capacity churn and irreversible information loss. The default GrowPage policy therefore favors conservative *Compress & Hold* and incremental growth, which provides more stable online capacity adaptation under PagedAttention-based serving.

I REQUEST-LEVEL CAPACITY ALIGNMENT

Figure 2 shows that the minimum sufficient fixed KV budget varies substantially across requests. We further examine whether GrowPage's online capacity allocation reflects this request-level demand heterogeneity. Specifically, we group requests according to the minimum sufficient budget $B_i^\star$ identified in Figure 2, and rerun the same requests with GrowPage using the same random seed. For each request, we record its average allocated KV capacity over the decoding trajectory. Figure 11 reports the resulting request-level capacities together with the mean and 95% confidence interval for each fixed-budget class.

Across the populated budget classes, requests requiring larger fixed budgets generally receive larger adaptive capacities from GrowPage. This trend is particularly clear on LiveCodeBench, where the average adaptive capacity increases consistently from the 512-token class to the 8K-token class. AMC23 shows the same tendency over its populated 512, 1K, and 2K classes. AIME24 also exhibits an overall positive relationship, although the sparsely populated 4K and 8K classes show larger variance. These results indicate that GrowPage does not allocate capacity uniformly across requests, but instead adapts its runtime KV allocation in accordance with their underlying memory demands.

Importantly, the adaptive capacity is not expected to numerically match $B_i^\star$. The latter denotes the minimum *fixed* capacity required throughout an entire generation, whereas GrowPage changes the allocated capacity online as demand evolves. Consequently, a request belonging to a high-budget class can still maintain a smaller average capacity because additional pages are acquired progressively as growth decisions occur. The default policy preserves the acquired capacity during subsequent compression operations, so the lower average capacity arises from deferred allocation rather than active capacity shrinkage.

We also observe that requests categorized as *Unresolved* tend to receive relatively large adaptive capacities. Since these requests are not consistently solved under any tested fixed budget, this behavior suggests that GrowPage responds to their elevated attention demand rather than to the eventual correctness outcome. Overall, the alignment between $B_i^\star$ and GrowPage's adaptive allocation provides request-level evidence that the proposed demand signal translates heterogeneous reasoning demands into differentiated KV capacity.