SELECT TWICE, FREEZE ONCE: JACOBIAN-INSPIRED KV CACHE SELECTION FOR MULTI-TURN & LONG-CONTEXT LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

The long-term deployment of multi-turn Large Language Model (LLM) systems depends on reliable and efficient handling of ever-extending context windows. As the key-value (KV) cache scales linearly with sequence length, it has become a primary bottleneck for memory and throughput. This overhead is particularly detrimental to multi-turn systems, which generate a high volume of prefill and decode calls and are therefore disproportionately sensitive to the overhead of low-rank approximation techniques. Existing KV cache selection methods rely on the attention map as the primary selection signal. We find that the attention-output Jacobian motivates considering local sensitivity beyond individual attention weights when selecting entries for reuse. We introduce PrescienceKV, which combines a linear surrogate derived from a quadratic scoring proxy with windowed attention-based reranking. The surrogate constructs candidates, while reranking aggregates local value sensitivity over recent prompt queries and the first decode query to select the final top-k historical subset. Both stages run at the first decode step of each turn over the prefilled history, after which the subset is frozen and all newly generated KV entries are appended without further selection. We evaluate PrescienceKV on RULER and LongBench across five models for accuracy and efficiency. PrescienceKV reduces the selected historical KV budget by over 96% at 128K context while maintaining accuracy comparable to full attention. At capacity, PrescienceKV sustains a decode batch of 16–20 across context lengths up to 124K, where full attention's maximum batch drops to 2, yielding decode-throughput speedups up to $6.01\times$. In the separate per-token latency comparison at 128K, it reduces latency by 33% relative to full attention.

1 INTRODUCTION

In recent years, the applications of LLMs have evolved from static tasks to complex multi-agent systems operating over extended periods. These systems require context windows spanning millions of tokens, necessitating effective memory management. The KV cache scales linearly with sequence length, and each autoregressive decoding step requires all previous key-value representations. Consequently, current methodologies are typically memory-bound (Tang et al., 2024; Sun et al., 2024; Kwon et al., 2023; Xiao et al., 2023; Zhang et al., 2023), constrained by the system's capacity to store and efficiently access the growing KV cache as context lengths scale. This memory bottleneck limits throughput and scalability in long-context inference scenarios (Sun et al., 2024). Furthermore, this problem is exacerbated in multi-agent, multi-turn, and other dynamic reasoning tasks where the context window size is not fixed.

Existing work mainly addresses these limitations by KV cache eviction or selection methodologies. KV cache eviction (Kwon et al., 2023; Xiao et al., 2023; Zhang et al., 2023; Li et al., 2024; Kariyappa & Suh, 2026) discards KV information during deployment based on specific relevance conditions. These methods often result in information loss and reduced accuracy, especially in multi-turn scenarios. Kariyappa & Suh (2026) address this by introducing a side-thread that computes long-term token relevance using a trained Large Reasoning Model, but this requires training, which introduces a significant overhead that we avoid.

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.

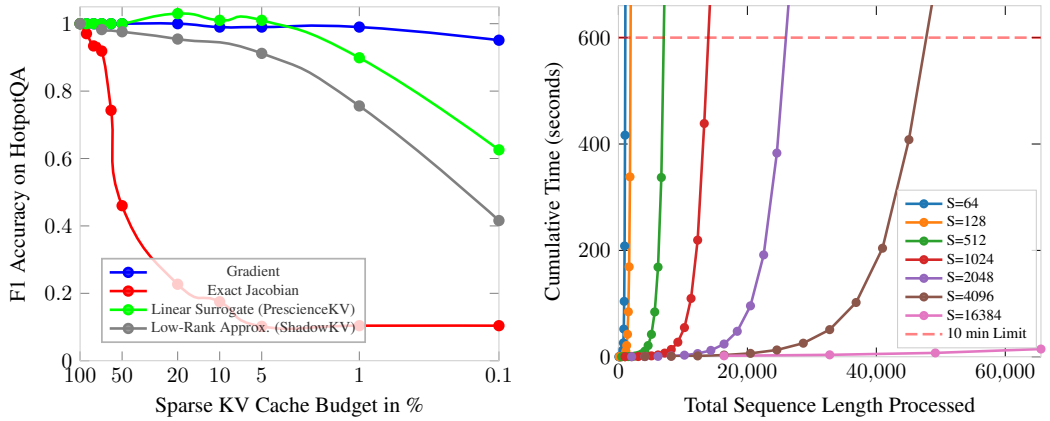

Figure 1: **Left:** KV selection methods on the HotpotQA dataset under varying KV budgets. Attention-based relevance estimation degrades under very small budgets, while Jacobian-based selection retains quality down to 0.1% of the original budget. This diagnostic uses the exact Jacobian and does not reflect the deployed surrogate's performance. ShadowKV (Sun et al., 2024) uses low-rank approximation for selection. **Right:** Cumulative prefill latency for ShadowKV (Sun et al., 2024) under incrementally growing context windows, representative of multi-turn settings (Wu et al., 2026; Kariyappa & Suh, 2026). The low-rank approximator is recomputed at each incremental prefill step.

KV cache selection (Tang et al., 2024; Lee et al., 2024; Sun et al., 2024; Liu et al., 2025b; Xu et al., 2026; Gao et al., 2026; Cheng et al., 2026; Liu et al., 2025c) methods maintain persistent KVs storage on HBM (Tang et al., 2024) or CPU (Lee et al., 2024; Sun et al., 2024; Liu et al., 2025b) and apply sparse attention to limit computational memory overhead. However, offloading to CPU introduces latency overhead during decoding. ShadowKV (Sun et al., 2024) reduces decoding latency via low-rank approximation during prefill, but this increases prefill latency, which is detrimental for multi-turn use cases (Wu et al., 2026), as shown in Figure 1. As shown in Figure 1, smaller context windows lead to repeated recomputation over previously processed tokens, resulting in higher cumulative latency. Additionally, the majority of existing work selects relevant keys and values based on the attention map. Our diagnostic in Figure 1 shows that attention weights alone do not identify the entries preferred by the output-sensitivity-based selection. Moreover, obtaining the attention map often requires additional computations or memory I/O overhead because modern inference engines, such as FlashAttention (Dao et al., 2022), do not expose it by default.

Consequently, efficient and reliable serving for single-turn, multi-turn, and long-context LLMs requires adherence to the following requirements: (i) reduced memory usage, (ii) minimized prefill and decode latency, (iii) maintained accuracy, (iv) ease of integration, and (v) dynamic offloading without a significant increase in latency. We argue that current methods do not sufficiently address this problem because they rely purely on selection via the attention map. Our analysis in Figure 1 indicates that the key-cache Jacobian describes directional output sensitivity through the queries and values, beyond individual attention weights. Inspired by this observation, we derive a cheap linear surrogate from a quadratic scoring proxy to construct candidates without evaluating the full Jacobian. We then aggregate attention over recent prompt queries and the first decode query, selecting entries with high local value sensitivity for at least one observed query. This window informs the final top-k subset before it is frozen for the turn, separating candidate construction from selection for reuse.

Our contributions are as follows:

- We analyze local attention-output sensitivity to keys and values, and provide an exact-Jacobian diagnostic of KV selection. This distinguishes value-dependent key sensitivity from attention weights, which exactly measure local value-Jacobian operator norms.
- We introduce an inexpensive linear surrogate by differentiating a quadratic feature-space proxy, retaining signed query-key directional agreement for candidate filtering. Our controlled attention-only ablation evaluates the benefit of this prefilter under matched budgets and reuse settings.

- We combine windowed surrogate filtering with attention reranking over recent prompt queries and the first decode query, then freeze the final top-k historical subset at step 0 of each turn. All newly generated KV entries are appended, avoiding repeated selection within the turn while reconstruction over the next turn's prefilled history allows the subset to adapt.
- We provide PrescienceKV, an algorithm that integrates these observations into a practical KV offloading and selection pipeline applicable to various Large Language Models. The goal of our algorithm is to demonstrate the utility of our observations in practical applications and to serve as inspiration for future work.
- We provide extensive analysis on varying models and datasets, demonstrating the effectiveness of our approach in reducing latency and memory-bandwidth while maintaining accuracy comparable to the full KV cache.

2 PRELIMINARIES

2.1 ATTENTION AND THE KV CACHE

Consider a transformer-based LLM with L layers, H_q query heads, and H_{kv} KV heads per layer. For a given head, let $\mathbf{Q} \in \mathbb{R}^{n_q \times d}$, $\mathbf{K} \in \mathbb{R}^{n_k \times d}$, and $\mathbf{V} \in \mathbb{R}^{n_k \times d}$ denote the query, key, and value matrices, where n_q is the number of query tokens, n_k is the number of key-value tokens, and d is the head dimension. The scaled dot-product attention is defined by the attention map $\mathbf{A} \in \mathbb{R}^{n_q \times n_k}$ as

$$\mathbf{Y} = \mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\Big((\mathbf{Q}\mathbf{K}^T)/\sqrt{d}\Big)\,\mathbf{V} = \mathbf{A}\mathbf{V}\,. \tag{1}$$

During autoregressive decoding, each new token attends over all previously computed keys and values. To avoid recomputation, these keys and values are stored in the KV cache, which grows linearly with the sequence length n_k.

2.2 PROBLEM FORMULATION

Given a historical KV cache of n_k tokens and a budget $b \ll n_k$ for its selected subset, the KV cache selection problem is to find an index subset $\mathcal{S} \subset \{1, \ldots, n_k\}$ with $|\mathcal{S}| \leq b$ such that

$$\mathbf{Y}_{\mathcal{S}} = \mathrm{Attention}(\mathbf{Q}, \mathbf{K}_{\mathcal{S}}, \mathbf{V}_{\mathcal{S}}) \approx \mathbf{Y}\,, \tag{2}$$

where $\mathbf{K}_{\mathcal{S}}$ and $\mathbf{V}_{\mathcal{S}}$ denote the rows of $\mathbf{K}$ and $\mathbf{V}$ indexed by $\mathcal{S}$. The goal is to minimize the approximation error $\|\mathbf{Y} - \mathbf{Y}_{\mathcal{S}}\|$ while keeping the selection cost low. In our per-turn policy, this budget applies to historical entries selected at step 0, with all subsequently generated KV entries appended outside this budget.

3 JACOBIAN-INSPIRED KV CACHE SELECTION

We first examine the local sensitivity of attention outputs to the key and value cache. This motivates two complementary selection stages: an inexpensive derivative-based surrogate constructs candidates, and windowed attention selects a historical subset for reuse. Both stages run at the first decode step of each turn, after which the final subset is frozen and all newly generated KV entries are appended.

3.1 JACOBIAN-INSPIRED RELEVANCE

The attention output's sensitivity to the key cache contains information beyond individual attention weights. For one query $\mathbf{q}$, let a_j be the attention weight of token j and $\mathbf{y} = \sum_j a_j \mathbf{v}_j$. The chain rule through the softmax provides the per-key block of the Jacobian $\mathbf{J}_{\mathbf{K}}$ as

$$\mathbf{J}_{\mathbf{k}_j} = \frac{\partial \mathbf{y}}{\partial \mathbf{k}_j} = \frac{a_j}{\sqrt{d}}(\mathbf{v}_j - \mathbf{y})\mathbf{q}^T\,. \tag{3}$$

The residual $\mathbf{v}_j - \mathbf{y}$ arises from the competitive softmax structure and the propagation of key perturbations through the values. Although the softmax Jacobian $\mathrm{diag}(\mathbf{a}) - \mathbf{a}\mathbf{a}^T$ is determined by the attention vector, the output Jacobian additionally depends on the values and the query direction. Figure 1 evaluates exact-Jacobian selection as a diagnostic, separately from the deployed surrogate.

For fixed queries and keys, the local value Jacobian has a simpler form:

$$\mathbf{J}_{\mathbf{v}_j} = \frac{\partial \mathbf{y}}{\partial \mathbf{v}_j} = a_j \mathbf{I}_d, \qquad \|\mathbf{J}_{\mathbf{v}_j}\|_2 = a_j \,. \tag{4}$$

Here $\|\cdot\|_2$ is the operator norm, so attention weights exactly measure local output sensitivity to value perturbations. These identities motivate complementary roles in our pipeline: a derivative-based proxy provides inexpensive candidate filtering, while windowed attention aggregates local value sensitivity for final selection. The proxy does not evaluate the full key Jacobian, and local sensitivity is not a guarantee of the effect of deleting a token or of its importance to future queries.

3.2 Linear Surrogate for Candidate Filtering

We construct a cheap derivative-based score rather than evaluating the full value-dependent key Jacobian. Using the second-order Taylor polynomial of the element-wise key exponential, we define a quadratic feature-space proxy,

$$F_2(\mathbf{q}, \mathbf{k}) = \sum_{\ell=1}^{d} e^{q_\ell} \left(1 + k_\ell + \tfrac{1}{2} k_\ell^2\right) \,. \tag{5}$$

Its derivative with respect to the keys is exactly

$$\nabla_{\mathbf{k}} F_2(\mathbf{q}, \mathbf{k}) = \exp(\mathbf{q}) \odot (\mathbf{1} + \mathbf{k}). \tag{6}$$

Summing its coordinates provides a score that measures alignment with the query-weighted direction $\exp(\mathbf{q})$. This construction is inspired by sensitivity-based relevance, but does not assume that $\exp(\mathbf{Q})\exp(\mathbf{K})^T$, which is used to obtain the Taylor polynomial, approximates softmax attention or that its derivative reproduces the softmax Jacobian. We use the score only to construct candidates for the subsequent attention-based selection. Appendix C provides the derivation and Appendix D compares alternative scoring orders empirically.

3.3 Windowed Selection for Reuse

We use two token-level scores to construct candidates, combining the signed derivative sum with normalized directional agreement:

$$T(\mathbf{q}, \mathbf{k}_j) = \mathbf{1}^T \nabla_{\mathbf{k}_j} F_2 = \sum_{\ell=1}^{d} e^{q_\ell} (1 + k_{j\ell}), \tag{7}$$

$$S(\mathbf{q}, \mathbf{k}_j) = \frac{\exp(\mathbf{q})^T (\mathbf{1} + \mathbf{k}_j)}{\|\exp(\mathbf{q})\|_2 \, \|\mathbf{1} + \mathbf{k}_j\|_2}. \tag{8}$$

At step 0 of each turn, the selection window $\mathcal{W}$ contains the current decode query and up to w_q recent prompt queries, for each layer and query head g. For each score branch, we compute

$$\overline{T}_{g,j} = \log \sum_{\tau \in \mathcal{W}} \exp\big(T(\mathbf{q}_{\tau,g}, \mathbf{k}_{h(g),j})\big) \,, \tag{9}$$

$$\overline{S}_{g,j} = \log \sum_{\tau \in \mathcal{W}} \exp\big(S(\mathbf{q}_{\tau,g}, \mathbf{k}_{h(g),j})\big) \,, \tag{10}$$

where $h(g)$ is the associated KV head. We apply the *Logsumexp* to smooth the maximum over the observed queries. The top b_c indices are selected separately for each branch and query head, then united with the last w_{rec} historical indices and o outlier indices to form the candidate set $\mathcal{C}$. Here, b_c is the candidate count per branch and query head before union, w_{rec} is the recent-token count, and o is the outlier count. The candidate count after union, $D = |\mathcal{C}|$, differs from the per-branch count b_c and the final budget b. We retain the full expressions for T and S, as the query-dependent constant in T cannot be dropped before window aggregation.

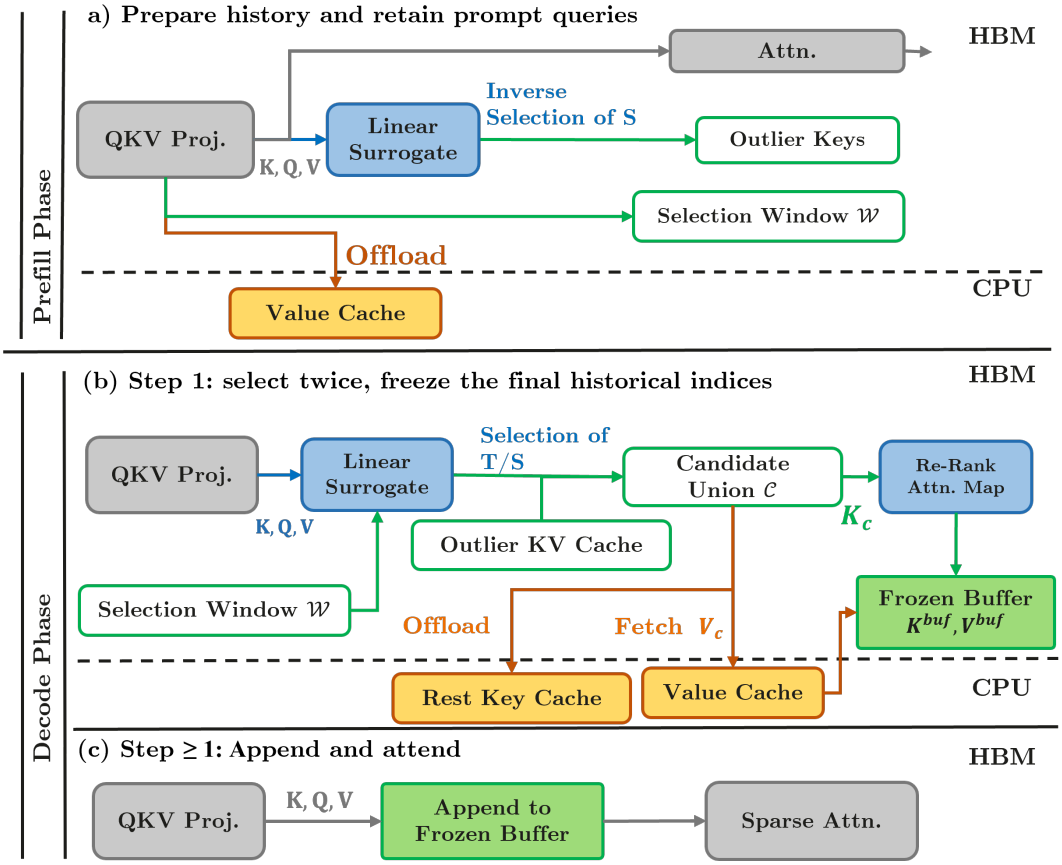

Figure 2: Overview of PrescienceKV, shown per layer. Prefill retains outliers and recent queries while offloading the remaining values. Stage 1 constructs the candidate tensors $\mathbf{K}_{\mathcal{C}}$ and $\mathbf{V}_{\mathcal{C}}$. Stage 2 selects the final top-b historical subset per KV head. In subsequent steps, the selected indices remain frozen, while newly generated entries are appended and attention weights change with each query. A new turn reconstructs the selection over the whole prefilled history.

We rerank the candidates using attention over the same window:

$$a^{\mathcal{C}}_{\tau,g,j} = \frac{\exp(\mathbf{q}^T_{\tau,g}\mathbf{k}_{h(g),j}/\sqrt{d})}{\sum_{i\in\mathcal{C}}\exp(\mathbf{q}^T_{\tau,g}\mathbf{k}_{h(g),i}/\sqrt{d})}, \qquad p_{h,j} = \max_{\tau\in\mathcal{W},\, g\in G(h)} a^{\mathcal{C}}_{\tau,g,j}, \tag{11}$$

where $G(h)$ groups the query heads sharing KV head h. The final historical subset is the top-b candidates under $p_{h,j}$, separately for each KV head. Stored prompt queries retain their original positional encoding and act as probes over the completed history without their original causal masks; generation attention remains causal. Once the candidate choices are fixed, $p_{h,j}$ tells us how large the sensitivity of the output can be, considering all positions in the window and all GQA groups. This favors entries with high local sensitivity for an observed query, using the window as a heuristic for relevance during reuse.

4 The PrescienceKV Algorithm

PrescienceKV, as seen in Figure 2, is divided into a prefill stage and a decoding stage. The **prefill** stage predicts an outlier KV cache, i.e., the subset of cache entries that have a high likelihood of being required but are hard to predict during decoding. To obtain this subset we use the relative-surrogate score S from Equation 8, rather than attention-based reranking. The resulting set retains entries with low relative-surrogate scores as potential outliers that the candidate filter might otherwise miss. All remaining values are offloaded to the CPU. We also retain up to w_q recent prompt queries per layer and query head, with their original positional encoding for step-0 selection.

At the first **decode** step of each turn, we combine the current query with the retained prompt queries to form the selection window. The token-level T and S scores are reduced over this window to identify the candidate indices $\mathcal{C}$, with corresponding keys $\mathbf{K}_{\mathcal{C}}$ and values $\mathbf{V}_{\mathcal{C}}$ in Figure 2. We then compute candidate-restricted attention for each window query, take the maximum over the window and associated query heads, and select the final top-b historical subset per KV head, overlapping reranking with any required value transfers.

We freeze the final subset selected by stage 2 and append all newly generated KV entries without pruning them during the turn. Subsequent steps recompute attention weights over this growing buffer, but do not repeat filtering or top-k selection. The first decode step of each new user turn reconstructs the selection over the whole prefilled history, allowing previously excluded entries to be selected again. The experiments evaluate this trade-off between selection overhead and accuracy.

5 Evaluation

We evaluate PrescienceKV in terms of accuracy, decode efficiency, and multi-turn retrieval. The appendix provides additional NIAH results, component ablations, and separate efficiency and memory evaluations.

5.1 Implementation Details

Our main LongBench and RULER evaluations cover five models: Llama-3.1-8B, Llama-3-8B-1M, Qwen2.5-3B, GLM-4-9B-1M, and Qwen3-8B. We compare against Quest (Tang et al., 2024), SnapKV (Li et al., 2024), InfiniGen (Lee et al., 2024), FreeKV (Liu et al., 2025a), and ShadowKV (Sun et al., 2024), using their provided implementations and recommended parameters. We evaluate on LongBench (Bai et al., 2024) (contexts up to 8,192 tokens), RULER (Hsieh et al., 2024) (nine tasks at 65,536 and 131,072 tokens), and NIAH (Kamradt, 2023) (appendix).

Selection budgets, query-window settings, decoding conventions, and context-length units are specified in Appendix A. All experiments use a single 80 GB accelerator. PrescienceKV is integrated into vLLM (Kwon et al., 2023) as a custom attention backend.

5.2 Efficiency

Our efficiency evaluation distinguishes single-stream latency from aggregate decode throughput at higher concurrency. In Table 10, we report single-batch latency on HotpotQA at 8K tokens and QA1 at 64K and 128K tokens, comparing full attention, ShadowKV, and PrescienceKV. Single-batch gains remain limited by the first-token overhead, which diminishes at larger context lengths and improves further with more concurrent users. We provide an extensive efficiency evaluation in Appendix K of the appendix.

In Table 1(a), we report the maximum decode batch each method can sustain at capacity across context lengths, along with the throughput speedup of PrescienceKV over full attention. Full attention's batch decreases from 16 to 2 as context grows due to KV cache memory pressure, while PrescienceKV maintains a batch of 16–20, yielding speedups from $1.24\times$ to $6.01\times$. Methods that offload to host memory (FreeKV Liu et al. (2025a), InfiniGen Lee et al. (2024)) can sustain larger batches but incur higher per-token latency, as shown in Table 1(b): at 128K context, PrescienceKV reduces per-token latency by 33% relative to full attention, while all other methods are 11–192% slower.

5.3 Multi-Turn Evaluation

We evaluate retrieval across conversation turns using multi-turn RULER NIAH and LoCoMo (Maharana et al., 2024). Figure 3 compares PrescienceKV with full attention and representative selection and eviction methods. On NIAH, PrescienceKV remains close to full attention over the four evaluated turns. On LoCoMo, its recall varies with the turn and sample, while SnapKV shows lower recall after the first turn in the displayed sequence.

Table 1: Efficiency comparison with a budget of 4,096 entries per KV head for Qwen3-8B on QA1. **(a)** Maximum decode batch and vLLM throughput speedup of PrescienceKV over full attention. Full vLLM, Ours vLLM, and Ours HF, ShadowKV, Quest batch sizes are measured, while other methods' capacities are derived from memory budgets. ^{H} denotes KV cache on HBM, ^{C} on CPU. **(b)** Per-token decode latency (ms) and ratio vs. full attention (lower is better). We provide implementations in Hugging Face Transformers (HF) and vLLM. Q, SK, and FK denote Quest, ShadowKV, and FreeKV, respectively.

(a) Max decode batch

Ctx	Full vLLM	Ours vLLM	Ours HF	Q HF	SK HF	FK HF	Spd vLLM
16K	16	20	**90**	15	64	25^{H}	1.24×
32K	8	20	**90**	7	32	22^{C}	2.12×
64K	4	**20**	**84**	3	16	11^{C}	**3.48**×
124K	2	**16**	**72**	1	8	6^{C}	**6.01**×

(b) Per-token latency (ms)

Method	64K	vs	128K	vs
Full	32.1	1.00	47.2	1.00
FreeKV	37.0	1.15	52.4	1.11
Quest	41.7	1.30	62.9	1.33
InfiniGen	80.3	2.50	137.9	2.92
ShadowKV	45.5	1.42	59.1	1.25
PrescienceKV	**30.7**	**0.96**	**31.6**	**0.67**

5.4 ACCURACY

We evaluate accuracy on LongBench and RULER at their respective sparse-attention budgets. Multi-turn retrieval is reported in Figure 3, with additional NIAH results in the appendix.

Repeated evaluations of PrescienceKV on RULER and LongBench, including runs with different decoding temperatures and random seeds, produced the same reported benchmark scores for the tested configurations. We therefore report point estimates without run-to-run error bars. This observation describes repeatability on the evaluated examples and does not quantify uncertainty arising from the finite benchmark sample.

LongBench (Bai et al., 2024) is evaluated on six tasks with contexts up to 8,192 tokens. In Table 2, PrescienceKV maintains accuracy comparable to full attention with a sparse-attention budget of 2,048 entries and without low-rank reconstruction. The controlled component ablation in Table 7 evaluates the contributions of surrogate filtering and outlier retention.

RULER (Hsieh et al., 2024) evaluates nine long-context tasks at 65,536 and 131,072 tokens. In Table 3, PrescienceKV uses 4,096 selected entries per KV head, corresponding to a 93.75% reduction at 64K and a 96.875% reduction at 128K, while achieving competitive accuracy without low-rank approximation. Our method incurs minimal latency increases and maintains stability across varying context lengths. Table 8 provides a separate budget ablation on LongBench in the appendix. Section M.1 evaluates prompt-query window size on eight RULER tasks at 64K, where 32 prompt queries improve the mean by about 1.2 percentage points over current-query-only selection in a fixed-request setting.

Table 2: LongBench at context lengths up to 8,192 tokens with a selected historical budget of 2,048 entries per KV head.

Model	WikiMQA	TQA	HQA	MQA	MSQ	NQA	Avg.
Llama-3.1-8B	35.91	18.68	46.31	47.27	21.12	21.22	31.75
Quest (Tang et al., 2024)	25.55	**19.81**	45.38	44.92	20.61	21.37	29.61
ShadowKV (Sun et al., 2024)	33.41	18.7	44.1	40.4	23.1	**22.6**	30.39
PrescienceKV (Ours)	**36.55**	18.8	**47.2**	**45.35**	**23.2**	**22.6**	**32.28**
Qwen2.5-3B	34.46	37.31	33.77	47.42	17.07	20.07	31.68
Quest (Tang et al., 2024)	13.32	34.58	30.8	38.12	14.27	10.64	23.62
ShadowKV (Sun et al., 2024)	34.63	34.6	36.7	45.8	18.4	20.5	31.77
PrescienceKV (Ours)	**34.73**	**40.31**	**41.17**	**46.61**	**18.85**	**20.58**	**33.71**

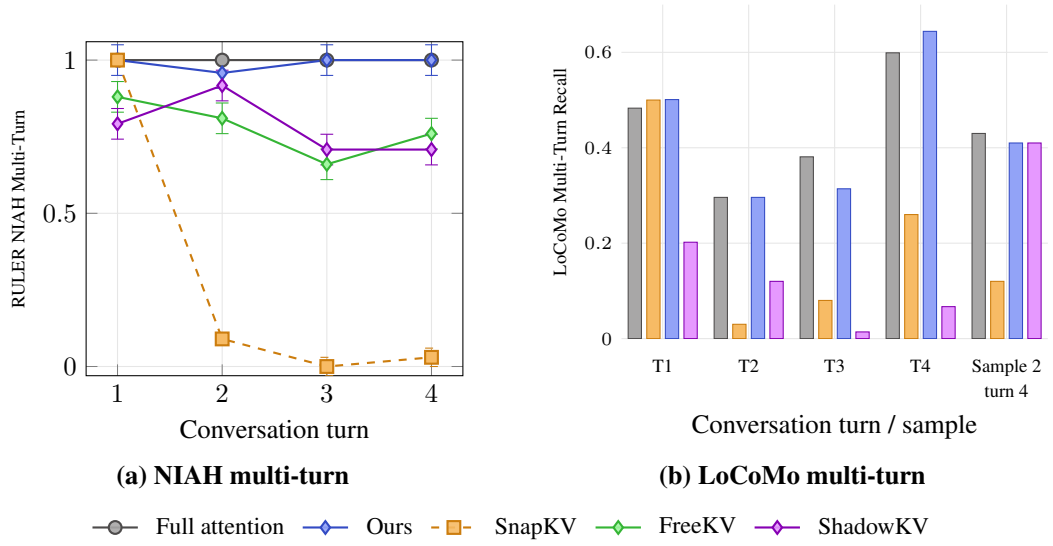

Figure 3: Multi-turn retrieval performance. (a) RULER NIAH multi-turn retrieval. (b) LoCoMo multi-session dialogue retrieval under a fixed KV budget. Eviction methods suffer from loss of information impacting the performance from the second conversation turn.

Table 3: RULER accuracy at context lengths of 65,536 and 131,072 tokens. All methods use a KV cache budget of 4,096 entries, yielding a reduction of up to 96%.

Model	Context Length: 65536 Tokens									
	S1	S2	MK2	MQ	MV	VT	FWE	QA-1	QA-2	Avg.
Llama-3-8B-1M	100.0	100.0	100.0	99.0	95.8	78.1	67.7	77.1	51.0	85.4
SnapKV (Li et al., 2024)	**100.0**	**100.0**	96.7	99.0	**98.4**	78.1	58.6	73.2	**51.0**	83.9
InfiniGen (Lee et al., 2024)	**100.0**	**100.0**	**100.0**	96.5	95.8	75.0	64.0	**80.9**	**51.0**	84.8
Quest (Tang et al., 2024)	**100.0**	**100.0**	**100.0**	99.0	**98.4**	78.1	62.2	73.2	47.9	84.3
FreeKV (Liu et al., 2025a)	**100.0**	**100.0**	96.7	98.1	95.8	78.1	60.4	**80.9**	44.7	83.9
ShadowKV (Sun et al., 2024)	**100.0**	**100.0**	99.0	95.8	90.1	**80.2**	74.0	74.0	49.0	84.7
PrescienceKV (Ours)	**100.0**	**100.0**	**100.0**	**99.7**	95.8	77.5	**74.3**	76.0	**51.0**	**86.1**
GLM-4-9B-1M	100.0	100.0	100.0	100.0	95.8	100.0	73.3	66.7	60.0	88.4
SnapKV (Li et al., 2024)	**100.0**	90.0	76.7	85.8	50.0	**100.0**	68.9	60.0	56.7	76.5
InfiniGen (Lee et al., 2024)	43.3	50.0	53.3	65.8	45.0	34.7	56.7	56.7	43.3	49.9
Quest (Tang et al., 2024)	**100.0**	**100.0**	**100.0**	99.2	96.7	**100.0**	68.9	60.0	53.3	86.5
FreeKV (Liu et al., 2025a)	**100.0**	96.7	96.7	97.5	89.2	**100.0**	68.9	**63.3**	53.3	85.1
ShadowKV (Sun et al., 2024)	**100.0**	**100.0**	**100.0**	**100.0**	93.3	**100.0**	**73.3**	60.0	53.3	86.7
PrescienceKV (Ours)	**100.0**	**100.0**	**100.0**	95.8	**97.5**	**100.0**	67.8	**63.3**	**60.0**	**87.2**
Qwen3-8B	100.0	100.0	70.0	95.8	98.3	96.0	83.3	46.7	43.3	81.5
Quest (Tang et al., 2024)	**100.0**	**100.0**	**76.7**	91.7	92.5	94.0	80.0	30.0	40.0	78.3
InfiniGen (Lee et al., 2024)	**100.0**	60.0	46.7	44.2	39.2	92.0	76.7	20.0	33.3	56.9
SnapKV (Li et al., 2024)	**100.0**	93.3	63.3	70.8	39.2	95.3	**82.2**	36.7	40.0	69.0
FreeKV (Liu et al., 2025a)	**100.0**	**100.0**	66.7	84.2	86.7	95.3	80.0	40.0	40.0	77.0
ShadowKV (Sun et al., 2024)	**100.0**	**100.0**	70.0	95.0	**100.0**	96.7	81.1	**43.3**	**43.3**	81.0
PrescienceKV (Ours)	**100.0**	**100.0**	**76.7**	**96.7**	**100.0**	**98.0**	80.0	**43.3**	**43.3**	**82.0**
Model	Context Length: 131072 Tokens									
Llama-3-8B-1M	100.0	100.0	100.0	99.2	95.0	83.3	71.1	70.0	53.3	85.8
InfiniGen (Lee et al., 2024)	**100.0**	**100.0**	96.7	97.5	**98.3**	78.7	66.7	66.7	**53.3**	84.2
Quest (Tang et al., 2024)	**100.0**	**100.0**	**100.0**	99.2	96.7	83.3	70.0	70.0	**53.3**	85.8
FreeKV (Liu et al., 2025a)	**100.0**	**100.0**	96.7	95.8	88.3	**84.0**	72.2	70.0	50.0	84.1
SnapKV (Li et al., 2024)	**100.0**	**100.0**	**100.0**	94.2	65.8	81.3	62.2	66.7	**53.3**	80.4
ShadowKV (Sun et al., 2024)	**100.0**	**100.0**	**100.0**	**100.0**	94.2	80.0	**76.7**	70.0	**53.3**	86.0
PrescienceKV (Ours)	**100.0**	**100.0**	**100.0**	**100.0**	96.7	83.3	73.3	**73.3**	**53.3**	**86.7**

6 Related Work

6.1 KV Cache Offloading & Selection

KV cache selection retains all KV values while selecting the most relevant ones at runtime, reducing effective sequence length without information loss. As the KV cache grows linearly, offloading can help manage device capacity, often requiring different fetching strategies. Quest (Tang et al., 2024) defines query-dependent KV pages, which are retrieved based on the highest attention scores approximated between each page. It uses these pages as selection criteria to fetch only the relevant values from the offloaded KV cache. InfiniGen (Lee et al., 2024) leverages the insight that a few important tokens can predict which tokens are essential for the next attention layer. It uses Singular Value Decomposition (SVD) and other projections to prefetch the most important KV components. ShadowKV (Sun et al., 2024) applies low-rank approximation to pre-RoPE keys and separately constructs landmarks and outliers from post-RoPE keys during prefill, while offloading the value cache. This design increases supported concurrency while limiting decoding overhead. Similar to ShadowKV, we offload only the value cache but do not require low-rank approximation. Methods relying on low-rank approximations, such as SVD, cause a significant increase in latency during prefill-intensive tasks, such as multi-turn systems, as shown in Figure 1. Additionally, ShadowKV and other introduced works use attention as the selection criterion. Our controlled attention-only ablation shows that surrogate-assisted selection can improve accuracy under the evaluated budget and reuse policy. Other approaches (Gao et al., 2026; Cheng et al., 2026; Liu et al., 2025c) also investigate relevance estimation for KV cache selection.

6.2 KV Compression & Eviction

KV cache compression improves runtime efficiency and reduces memory consumption. KVTC (Staniszewski & Łańcucki, 2026) and, in parallel, TurboQuant (Zandieh et al., 2026) propose transform coding schemes that reduce persistent KV storage. KVTC uses signal-dependent low-rank approximation, which adds preprocessing work when applied repeatedly in multi-turn settings, a challenge that also occurs with ShadowKV (Sun et al., 2024). While KV compression improves runtime efficiency, it does not reduce the number of computations, and, heavy compression often suffers from either increased latency or accuracy degradation (Staniszewski & Łańcucki, 2026).

Token eviction (Kwon et al., 2023; Zhang et al., 2023; Xiao et al., 2023; Li et al., 2024) removes tokens to reduce persistent KV cache to a fixed size. Common inference engines such as vLLM (Kwon et al., 2023) deploy a simple least-recently-used strategy, which incurs significant performance degradation in long-context and multi-turn systems. StreamingLLM (Xiao et al., 2023) retains attention sinks alongside the most recent KV pairs to generalize over infinitely long sequences, thereby enabling eviction during deployment. SnapKV (Li et al., 2024) reduces storage overhead by using a local window of prompts to extract a set of prompt attention features most relevant for future processing. The core limitation is the known loss of information and performance degradation, particularly in multi-turn scenarios, as noted by Kariyappa & Suh (2026) and Sun et al. (2024).

7 Conclusion

We present PrescienceKV, an LLM serving methodology for reducing memory-bandwidth requirements in multi-turn inference systems. PrescienceKV is built on two key insights. First, low-rank approximation is not necessary for KV cache reduction and degrades latency, particularly in multi-turn settings. Second, local attention-output sensitivity motivates combining a derivative-based candidate score with windowed attention for selection before reuse. Our linear surrogate is derived from a quadratic scoring proxy, while attention-based reranking aggregates local value sensitivity across recent prompt queries and the first decode query. We freeze the final top-k historical subset within each turn and append all newly generated KV entries, avoiding repeated selection during decoding. Both stages reconstruct the selection over the whole prefilled history at each new turn's first decode step. Our experiments demonstrate efficiency improvements while maintaining accuracy comparable to the full KV cache in the evaluated settings.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 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.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Yifei Gao, Lei Wang, Rong-Cheng Tu, Qixin Zhang, Jun Cheng, and Dacheng Tao. Near-oracle kv selection via pre-hoc sparsity for long-context inference. *arXiv preprint arXiv:2602.08329*, 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Gregory Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main, 2023.

Sanjay Kariyappa and G Edward Suh. Sidequest: Model-driven kv cache management for long-horizon agentic reasoning. *arXiv preprint arXiv:2602.22603*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 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.

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.

Guang-Da Liu, Cheng-Wei Li, Zhenyu Ning, Jing Lin, Yiwu Yao, Dan-Ning Ke, Min-Yi Guo, and Jie-Ru Zhao. Freekv: Boosting kv cache retrieval for efficient llm inference. *ArXiv*, abs/2505.13109, 2025a. URL https://api.semanticscholar.org/CorpusID:278769562.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025b.

Songtao Liu, Hongwu Peng, Zhiwei Zhang, Zhengyu Chen, and Yue Guo. Multi-head low-rank attention. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025c. URL https://openreview.net/forum?id=lD13399d8J.

Adyasha Maharana, Dong-Ho Lee, S. Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *ArXiv*, abs/2402.17753, 2024. URL https://api.semanticscholar.org/CorpusID:268041615.

Konrad Staniszewski and Adrian Łańcucki. KV cache transform coding for compact storage in LLM inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=aNVKROYpLB.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Yongtong Wu, Shaoyuan Chen, Yinmin Zhong, Rilin Huang, Yixuan Tan, Wentao Zhang, Liyue Zhang, Shangyan Zhou, Yuxuan Liu, Shunfeng Zhou, et al. Dualpath: Breaking the storage bandwidth bottleneck in agentic llm inference. *arXiv preprint arXiv:2602.21548*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Jiaming Xu, Jiayi Pan, Hanzhen Wang, Yongkang Zhou, Jiancai Ye, Yu Wang, and Guohao Dai. Specontext: Enabling efficient long-context reasoning with speculative context sparsity in llms. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1832–1847, 2026.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

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.

AI USE STATEMENT

In this work, we used generative AI tools for assistance with code implementation and manuscript refinement. We have not used generative AI tools for data generation, experimental design, or result analysis. All experimental results were produced by the authors and verified for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide pseudocode for both the prefill and decode phases of PrescienceKV in the appendix (Algorithms 1 and 2). The implementation details in Section 5 specify the models, datasets, and hardware used, and Appendix A provides the selection budgets and remaining parameter settings. We will release the full implementation, including the vLLM integration, upon acceptance.

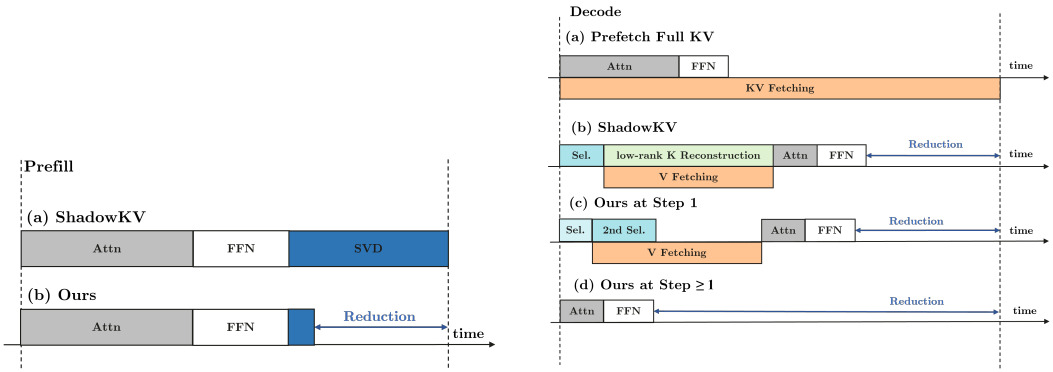

Figure 4: **Left:** Schematic comparison of pipeline architectures with and without SVD-based low-rank approximation. **Right:** Four schematic decode paths, from top to bottom: full-cache fetching, low-rank key reconstruction with value fetching, PrescienceKV at step 0 with two-stage selection and value fetching, and subsequent PrescienceKV steps that append new entries and attend to the retained buffer without reselection or historical-value fetching. Bar lengths are illustrative, not measured latency ratios; they do not imply that sparse decoding is faster than on-device full attention at batch one.

This appendix provides supplementary material to the main paper.

We first provide implementation settings and compare PrescienceKV with ShadowKV and Quest, then provide the surrogate derivation, order analysis, complexity analysis, and pseudocode. We next discuss limitations and future work, followed by component ablations and additional NIAH results. Finally, we report detailed efficiency and memory measurements and selection-schedule ablations.

A Implementation Details

We define the parameter required to reproduce the provided set of results. The final historical budget is $b = 2{,}048$ entries per KV head for LongBench and $b = 4{,}096$ for RULER. We conduct all experiments with $o = 64$ outliers and $w_{\text{rec}} = 128$ most recent historical entries in the candidate union. Unless varied in an ablation, the prefilter factor is $\gamma = \texttt{grad_factor} = 1.5$, giving $b_c = 1.5b$ candidates per score branch and query head before union, capped by the available history. Accuracy evaluations use request batch size $B = 1$, and set the prompt-query count to $w_q = 32$. These are selection budgets, not resident-memory measurements. We report memory components separately in Appendix L. Finally, for RULER, we use $\mathrm{K} = 1{,}024$ tokens, so 64K, 124K, and 128K denote 65,536, 126,976, and 131,072 tokens, respectively, following the evaluation format of ShadowKV.

B Detailed Method Comparison

We provide a detailed comparison of PrescienceKV against the two primary baselines evaluated in this work: ShadowKV (Sun et al., 2024) and Quest (Tang et al., 2024). Table 4 summarizes the key differences.

ShadowKV (Sun et al., 2024). ShadowKV computes a low-rank approximation of pre-RoPE keys during prefill using SVD and separately constructs landmarks and outliers from post-RoPE keys. At each decode step, it scores landmarks, reconstructs the selected keys, and retrieves the corresponding values. In our comparison with repeated prefill over growing histories, SVD is recomputed at each prefill step, increasing cumulative latency (Figure 1, right). PrescienceKV avoids low-rank approximation entirely, using the linear surrogate instead, and freezes the final historical KV subset selected by the second stage at the first decode step of each turn.

Table 4: Direct comparison of PrescienceKV with ShadowKV and Quest across key design dimensions.

	ShadowKV	Quest	PrescienceKV
Selection signal	Attention map	Attention map	Linear surrogate + windowed attention
Low-rank approx.	Yes (SVD)	No	No
Re-selects per step	Yes	Yes	No (stage-2 set frozen)
Prefill overhead	SVD computation	Chunk indexing	Outlier selection + query retention
Value offloading	Yes	No (default)	Yes
Historical selection	Query-dependent	Query-dependent	Rebuilt each turn

Quest (Tang et al., 2024). Quest partitions the key cache into chunks and selects relevant chunks per query based on approximated attention scores. The chunk selection is repeated at every decode step, yielding $\mathcal{O}(n_k/c \cdot d + b \cdot d)$ per step. Quest does not offload values by default, limiting its applicability in memory-constrained settings. PrescienceKV applies the linear surrogate and attention-based reranking at the first decode step of each turn, then reuses the final stage-2 historical subset without repeating selection. All newly generated KV entries are appended to the retained buffer, and the first decode step of the next user turn reconstructs the selection over the whole prefilled history.

Key distinction. PrescienceKV combines a derivative-based proxy for candidate construction with windowed attention for final selection, rather than evaluating the full key Jacobian during inference. At step 0, both stages use recent prompt queries and the current decode query, and plain top-k reranking determines the historical subset to freeze. The reranking score aggregates local value-Jacobian operator norms as described in Section 3.3. Subsequent queries recompute attention weights over the retained subset and all appended entries, not a new top-k selection from the stage-1 candidates. Per-turn reconstruction allows previously excluded history to be selected again. Appendix E accounts for reconstruction at each turn and growth of the attended buffer during generation.

C In-Depth Derivation of the Linear Surrogate

We distinguish the construction of the scoring proxy from the interpretation of the attention reranker. The feature product $F(\mathbf{q}, \mathbf{k}) = \sum_{\ell=1}^{d} e^{q_\ell} e^{k_\ell}$ defines a scoring function. Replacing each key exponential by its second-order Taylor polynomial defines F_2 in Equation 5. For fixed $\mathbf{q}$, differentiation gives

$$\frac{\partial F_2(\mathbf{q}, \mathbf{k}_j)}{\partial k_{j\ell}} = e^{q_\ell}(1 + k_{j\ell}), \qquad T(\mathbf{q}, \mathbf{k}_j) = \sum_{\ell=1}^{d} e^{q_\ell}(1 + k_{j\ell}). \tag{12}$$

The derivative is exact for the polynomial proxy. The signed sum is its directional derivative along $\mathbf{1}$, not a gradient norm, and costs $\mathcal{O}(d)$ per query-key pair. The normalized branch S in Equation 8 instead measures cosine similarity between $\exp(\mathbf{q})$ and $\mathbf{1} + \mathbf{k}_j$.

For one fixed query, $T(\mathbf{q}, \mathbf{k}_j) = \sum_\ell e^{q_\ell} + \exp(\mathbf{q})^T \mathbf{k}_j$, so the first term does not affect the raw ranking across keys. It must nevertheless be retained in the windowed construction: different queries have different offsets, which change their relative contributions inside logsumexp. The key-dependent denominator in S also prevents treating constant removal as a general ranking invariance. Both branches are evaluated in their original form before the separate window reductions in Equation 10 and Equation 9. Logsumexp is evaluated stably by subtracting the largest input before exponentiation.

We also evaluated the selection pipeline with and without normalization and obtained close to identical task scores in the tested configurations. This is an empirical observation, not justification for dropping a softmax derivative or for removing the normalized S branch from the described method. The order analysis in Appendix D concerns the usefulness of alternative scores, not a bound on their distance from the full attention Jacobian.

Table 5: End-to-end comparison of selection scores from different Taylor $\mathbf{T}$ expansions and the derivative-based linear surrogate.

Selection score	Llama-3-8B-1M		Qwen3-8B	
	acc	vs. base	acc	vs. base
$\nabla_{\mathbf{K}}\mathbf{T}_2$	0.667	(reference)	0.333	(reference)
$\mathbf{T}_2$	0.700	$+0.033 \pm 0.033$	0.367	$+0.033 \pm 0.076$
$\mathbf{T}_3$	0.700	$+0.033 \pm 0.033$	0.300	-0.033 ± 0.058
exp	0.667	$+0.000 \pm 0.000$	0.333	$+0.000 \pm 0.068$

Windowed local value sensitivity. Fix the candidates $\mathcal{C}$, query $\mathbf{q}_{\tau,g}$, and keys, and consider the candidate-restricted output $\mathbf{y}^{\mathcal{C}}_{\tau,g} = \sum_{i\in\mathcal{C}} a^{\mathcal{C}}_{\tau,g,i}\mathbf{v}_{h(g),i}$. For each retained candidate,

$$\frac{\partial \mathbf{y}^{\mathcal{C}}_{\tau,g}}{\partial \mathbf{v}_{h(g),j}} = a^{\mathcal{C}}_{\tau,g,j}\mathbf{I}_d, \qquad p_{h,j} = \max_{\tau\in\mathcal{W},\, g\in G(h)} \left\| \frac{\partial \mathbf{y}^{\mathcal{C}}_{\tau,g}}{\partial \mathbf{v}_{h,j}} \right\|_2 . \tag{13}$$

This equality explains the local sensitivity aggregated by the reranker. It holds with candidate membership fixed and does not differentiate through top-k selection. It is not the gradient of a sequence loss, which would also involve upstream derivatives, nor a guarantee about deleting entries and renormalizing attention. The prompt-query window supplies observed directions for selection before reuse, and its relevance to future queries is evaluated through the complete pipeline.

D Surrogate Order Analysis

The relevance score is the signed coordinate sum of the derivative of the quadratic proxy, which yields a linear surrogate. Table 5 compares the alternative Taylor truncation orders in an end-to-end evaluation. The second-order expansion achieves slightly higher accuracy point estimates, while the derivative-based score avoids its quadratic key term. The third-order expansion and full exponential do not consistently improve accuracy across the two evaluated models. These results describe an empirical accuracy–arithmetic trade-off, rather than establishing that higher-order terms contain no additional information or that differentiation always improves accuracy.

E Complexity Analysis

Let n_r denote the length of the whole history prefilled for turn r, d the head dimension, b the budget for the final historical KV subset selected by stage 2, and b'_r the number of stage-1 candidates scored at reconstruction. The selection window has $m_r = 1 + \min(w_q, n_r)$ query rows per query head, comprising the first decode query and up to w_q retained prompt queries. At decode step t, let $g_{r,t}$ be the number of new KV entries appended during the turn through that step. Attention then processes at most $b + g_{r,t}$ entries, not the larger stage-1 candidate set. For one new entry per step, $g_{r,t} = t + 1$ with $t = 0$ at the first decode step. Table 6 compares the arithmetic cost per query head, treating the GQA group size as fixed and excluding top-k, indexing, and data transfers. Quest uses chunks of size c.

Table 6: Per-decode-step arithmetic cost within a turn. PrescienceKV runs both selection stages at the first decode step of every turn and freezes the final historical subset selected by stage 2. Subsequent steps compute attention over this subset and all appended entries without repeating selection.

Method	First decode of each turn	Subsequent decodes
Full attention	$\mathcal{O}(n_r d)$	$\mathcal{O}((n_r + g_{r,t})d)$
Quest (Tang et al., 2024)	$\mathcal{O}((n_r/c + b)d)$	$\mathcal{O}(((n_r + g_{r,t})/c + b)d)$
PrescienceKV	$\mathcal{O}((m_r(n_r + b'_r) + b)d)$	$\mathcal{O}((b + g_{r,t})d)$

Full attention processes the entire cache at every decode step, including newly appended entries. Quest (Tang et al., 2024) instead computes a bound for each chunk and selects chunks for sparse attention, repeating this selection for every query.

ShadowKV (Sun et al., 2024) scores landmarks and reconstructs selected keys from low-rank factors at each decode step. Its cost depends on the landmark count, reconstruction rank, and cache hits, so the rank alone does not specify its selection cost. We discuss these operations separately rather than equating the low-rank dimension with the number of landmarks in the table.

At reconstruction, the original token-level T/S scores are evaluated over the window at cost $\mathcal{O}(m_r n_r d)$, and stage-2 attention scoring over the candidates costs $\mathcal{O}(m_r b'_r d)$. Logsumexp over window rows and the maximum over window rows and associated query heads do not change these leading arithmetic costs. The analysis describes the token-level scorer, not a chunk-landmark replacement. Both stages incur top-k and gathering costs at reconstruction, and fetching f_r nonresident candidate values transfers $\mathcal{O}(f_r d)$ elements. After stage 2, the selected historical KV subset is retained and all new KV entries are appended to its buffer. Subsequent steps only recompute attention weights and outputs over that buffer, costing $\mathcal{O}((b + g_{r,t})d)$ with no further surrogate scan or top-k reranking.

Over N_r decode steps in turn r, the total arithmetic cost is

$$\mathcal{O}\left(m_r(n_r + b'_r)d + N_r bd + d \sum_{t=0}^{N_r-1} g_{r,t} \right). \tag{14}$$

When all new entries are appended without within-turn pruning, the sum is $N_r(N_r + 1)/2$, giving an amortized cost of $\mathcal{O}(m_r(n_r + b'_r)d/N_r + (b + N_r)d)$ per step. For responses that are short relative to b, buffer growth is a small part of the repeated work, and the subsequent-decode cost is approximately $\mathcal{O}(bd)$. For longer responses, the appended entries must be included rather than assuming a constant-size attention buffer.

Across multiple turns, both selection stages are repeated over the history prefilled for each request, contributing $\mathcal{O}(d \sum_r m_r(n_r + b'_r))$ in addition to the per-token attention costs above. Prefill computation is separate from these decode costs. Access to the whole history at a new turn does not require retaining the full historical KV cache on-device between turns. Retained prompt queries require $\mathcal{O}(m_r d)$ storage per layer and query head at reconstruction. Materializing all surrogate and attention scores requires $\mathcal{O}(m_r n_r + m_r b'_r)$ temporary elements per query head; these reductions can instead be implemented in tiles, but their workspace must still be accounted for. Candidate K/V storage is temporary, while the retained decode KV buffer occupies $\mathcal{O}((b + g_{r,t})d)$ elements per KV head.

F ALGORITHM

To enhance reproducibility, we provide pseudocode separated into prefill and decoding stages. Algorithm 1 details outlier retention, value offloading, and prompt-query retention during prefill. Algorithm 2 describes step-0 selection and subsequent attention over the retained buffer. At reconstruction, the candidate union also includes the last w_{rec} historical entries. When gathering candidate values, we reuse values already resident on HBM rather than fetching them again from the CPU. At the first decode step of each turn, the current query and retained prompt queries form the selection window. Stage 1 constructs token-level candidates from the whole prefilled history using windowed T/S scores, and stage 2 selects the final top-b historical subset using windowed attention. The resulting $\mathbf{K}^{\text{buf}}$ and $\mathbf{V}^{\text{buf}}$ buffers retain that subset, with all newly generated KV entries appended during the turn. Attention is computed over these growing buffers at every step, but neither selection stage is repeated until reconstruction.

Here, B is the request batch size, n is the prefilled history length, $m = 1 + \min(w_q, n)$ is the selection-window size, and $D = |\mathcal{C}|$ is the temporary candidate count after union. Logsumexp reduces only the window axis, each attention row is normalized over candidates, and the final maximum reduces both the window and the query heads sharing a KV head. Selection queries keep their original positional encoding but are rescored without their original causal masks over the completed history. The final attention call uses the current query and only causally available KV entries. The candidate tensors $\mathbf{K}_{\mathcal{C}}$ and $\mathbf{V}_{\mathcal{C}}$ contain original cache entries, not gradients or transformed features. Candidate values are assembled on HBM in the same token-index order as the keys before final gathering into $\mathbf{K}^{\text{buf}}$ and $\mathbf{V}^{\text{buf}}$. After the final subset is selected, the attended buffer grows with every appended KV entry, without a rolling window or a repeated top-k operation. The per-turn flag is set again after the whole history is prefilled for a new user request, so both selection stages reconstruct the historical subset rather than filtering only the previous buffer.

Algorithm 1 PrescienceKV Prefill Phase

Input: $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{B \times H_{kv} \times n \times d}$, $\mathbf{Q} \in \mathbb{R}^{B \times H_q \times n \times d}$, outlier count o, prompt-query count w_q

$\mathbf{Q}^{\text{prompt}} \leftarrow \text{LastQueries}(\mathbf{Q}, \min(w_q, n))$ ▷ retain per layer

Compute linear surrogate

$\mathbf{Q}_s \in \mathbb{R}^{B \times H_q \times n \times d} \leftarrow \text{Exp}(\mathbf{Q})$

$\mathbf{K}_s \in \mathbb{R}^{B \times H_q \times n \times d} \leftarrow \text{Add}(1, \text{Repeat}(\mathbf{K}, H_q / H_{kv}))$

$\mathbf{G}_A \in \mathbb{R}^{B \times H_q \times n \times d} \leftarrow \text{Mult}(\mathbf{Q}_s, \mathbf{K}_s)$

Compute absolute relevance from Equation 7

$\mathbf{T} \in \mathbb{R}^{B \times H_q \times n} \leftarrow \text{Sum}(\mathbf{G}_A, \text{dim} = -1)$

Compute relative relevance from Equation 8

$\mathbf{S}_h \in \mathbb{R}^{B \times H_q \times n} \leftarrow \text{Divide}(\mathbf{T}, \text{Mult}(\text{Norm}(\mathbf{Q}_s), \text{Norm}(\mathbf{K}_s)))$

Obtain outlier from relative relevance

$\mathbf{I}^{out} \in \mathbb{R}^{B \times o} \leftarrow \text{ArgTopK}(\mathbf{S}_h, \text{largest} = \text{False})$

$\mathbf{V}^{out} \in \mathbb{R}^{B \times H_{kv} \times o \times d} \leftarrow \text{Gather}(\mathbf{V}, \mathbf{I}^{out})$

Offload the rest of the values to the CPU

$\mathbf{V}^{\text{CPU}} \leftarrow \mathbf{V} \setminus \{\mathbf{V}^{out}\}$

Compute prefill attention

Algorithm 2 PrescienceKV Decode Phase

Input: prefilled history $\mathbf{K}, \mathbf{V}$ at reconstruction, current query $\mathbf{Q}$, retained prompt queries $\mathbf{Q}^{\text{prompt}}$, new entries $\mathbf{K}^{new}, \mathbf{V}^{new}$, recent count w_{rec}, per-branch candidate count b_c, final budget b

State: retained buffers $\mathbf{K}^{\text{buf}}, \mathbf{V}^{\text{buf}}$ and outlier indices $\mathbf{I}^{out}$

Convention: history excludes the current step's new entries. Shapes below omit the request batch dimension.

if first decode step of the current turn **then**

$\mathbf{Q}^W \in \mathbb{R}^{H_q \times m \times d} \leftarrow \text{ConcatQueries}(\mathbf{Q}^{\text{prompt}}, \mathbf{Q})$

Stage 1: original token-level scores, then window reduction

$\mathbf{U} \leftarrow \exp(\mathbf{Q}^W)$, $\mathbf{K}^+ \leftarrow \mathbf{1} + \text{RepeatKVHeads}(\mathbf{K})$

$\mathbf{T} \in \mathbb{R}^{H_q \times m \times n} \leftarrow \mathbf{U}(\mathbf{K}^+)^T$

$\mathbf{S} \leftarrow \mathbf{T} / (\|\mathbf{U}\|_2 \|\mathbf{K}^+\|_2)$ ▷ pairwise row norms

$\overline{\mathbf{T}} \leftarrow \text{LogSumExp}(\mathbf{T}, \text{window})$, $\overline{\mathbf{S}} \leftarrow \text{LogSumExp}(\mathbf{S}, \text{window})$

$\mathbf{I}_T \leftarrow \text{ArgTopK}(\overline{\mathbf{T}}, b_c)$, $\mathbf{I}_S \leftarrow \text{ArgTopK}(\overline{\mathbf{S}}, b_c)$ ▷ per query head

$\mathcal{C} \leftarrow \text{UniqueAcrossHeads}(\mathbf{I}_T, \mathbf{I}_S, \text{LastIndices}(\mathbf{K}, w_{\text{rec}}), \mathbf{I}^{out})$

$\mathbf{K}_{\mathcal{C}} \leftarrow \text{IndexSelect}(\mathbf{K}, \mathcal{C})$

$\mathcal{R} \leftarrow$ candidate indices with values already resident on HBM

$\mathbf{V}^{\text{fetch}} \leftarrow \text{AsyncLoad}(\mathbf{V}^{\text{CPU}}, \mathcal{C} \setminus \mathcal{R})$

Stage 2: unmasked selection probes; softmax over candidates

$\mathbf{A} \in \mathbb{R}^{H_q \times m \times D} \leftarrow \text{Softmax}_{\text{candidates}}(\mathbf{Q}^W \text{RepeatKVHeads}(\mathbf{K}_{\mathcal{C}})^T / \sqrt{d})$

$\mathbf{P} \in \mathbb{R}^{H_{kv} \times D} \leftarrow \text{Max}_{\text{window,GQA group}}(\mathbf{A})$

$\mathbf{I}_A \leftarrow \text{ArgTopK}(\mathbf{P}, b)$ ▷ plain top-b per KV head

$\mathbf{V}_{\mathcal{C}} \leftarrow \text{MergeInIndexOrder}(\mathcal{C}, \text{Wait}(\mathbf{V}^{\text{fetch}}), \mathbf{V}^{\text{resident}}[\mathcal{R}])$

$\mathbf{K}^{\text{buf}} \leftarrow \text{Gather}(\mathbf{K}_{\mathcal{C}}, \mathbf{I}_A)$, $\mathbf{V}^{\text{buf}} \leftarrow \text{Gather}(\mathbf{V}_{\mathcal{C}}, \mathbf{I}_A)$

end if

Every decode step: append all new entries without selection

$\mathbf{K}^{\text{buf}} \leftarrow \text{Append}(\mathbf{K}^{\text{buf}}, \mathbf{K}^{new})$

$\mathbf{V}^{\text{buf}} \leftarrow \text{Append}(\mathbf{V}^{\text{buf}}, \mathbf{V}^{new})$

$\mathbf{Y} \leftarrow \text{Attention}(\mathbf{Q}, \mathbf{K}^{\text{buf}}, \mathbf{V}^{\text{buf}})$

G LIMITATIONS

Surrogate selection quality. The linear surrogate is derived from a quadratic feature-space proxy, not from an exact evaluation of the value-dependent key Jacobian. Its candidate rankings can therefore differ from rankings based on full output sensitivity. Our evaluations establish its usefulness in the tested pipeline, but do not provide a guarantee relating its rankings to full-Jacobian selection or token-removal error.

Candidate-budget choice. The prefilter count b_c controls the trade-off between candidate coverage and reconstruction cost. We vary this count in Table 15, but different models or workloads may favor a different setting from the default in Appendix A. This is a top-k budget choice, not a separate score threshold.

Local rather than loss sensitivity. Windowed attention measures local value sensitivity with queries, keys, and candidate membership fixed. It does not include upstream loss gradients or the downstream effects of cache selection across layers and generation steps.

Frozen historical selection within a turn. The first decode query and recent prompt queries provide observed directions for selection, but need not represent every query encountered later in the response. Attention weights adapt to each query over the retained buffer, but historical entries excluded by either selection stage cannot be recovered until the next reconstruction. The first decode step of each new user turn reconstructs the final subset over the whole prefilled history, but substantial query changes within a turn may still require more frequent reconstruction. Retaining all newly generated KV entries also increases the attended buffer size and attention cost over long responses, as described in Appendix E.

H FUTURE WORK

The attention Jacobian as a relevance signal is not specific to KV cache selection and may benefit related problems such as token eviction, speculative decoding, and context distillation. Characterizing when the surrogate and query window preserve useful candidates is a natural next step. Extending the per-turn reconstruction policy with additional within-turn refreshes could address scenarios where query drift makes the frozen historical subset stale. Adopting a holistic view of the selection pipeline rather than a layer-by-layer approach could improve the ability to capture long-range dependencies.

I ABLATION

Finally, we provide an ablation study on the individual components of PrescienceKV to demonstrate their relevance for multi-document retrieval on the LongBench dataset.

In Table 7, we compare our method against a baseline that uses only the attention map for selection. The attention-only baseline replaces the surrogate-based candidate score with attention scores while keeping the query window, candidate budget, final attention budget, outlier policy, and reuse schedule unchanged. We evaluate these configurations with a sparse budget of 2048 on the LongBench dataset. Additionally, we assess the inclusion of predicted outliers.

Table 7: Component ablation on Llama-3.1-8B with a budget of 2,048 entries per KV head. We compare surrogate-assisted selection with attention-only selection and remove predicted outliers separately.

Configuration	HQA	MQA	MSQ	NQA	Avg.
Full attention	46.31	**47.27**	21.12	21.22	33.98
Attn. Map Only	12.63	19.85	7.5	13.21	13.30
PrescienceKV without outliers	38.53	37.61	19.03	**23**	29.54
PrescienceKV	**47.2**	45.35	**23.2**	22.6	**34.59**

Table 8: **Ablation study** LongBench 2wikimqa, Qwen3-8B, 32K context, 178 prompts, greedy decoding. Paired differences against full attention on identical prompts; * marks > 2 SEM.

Method	$b = 256$	$b = 512$	$b = 1024$	$b = 2048$
Full attention	0.393	0.393	0.393	0.393
Ours	0.389	0.387	0.386	0.388
Paired difference				
Ours	-0.004 ± 0.010	-0.006 ± 0.007	-0.007 ± 0.006	-0.005 ± 0.006



Figure 5: Evaluation for the NIAH dataset for various context windows up to 128K tokens.

J Needle In A Haystack Evaluation

Needle In A Haystack (NIAH) (Kamradt, 2023) evaluates a model's ability to retrieve a specific piece of information from a long context filled with distractors, serving as a standard benchmark for long-context retrieval. Due to computational constraints, we assess our method's capability to process information across various context windows up to 128K tokens. We present these results for Qwen2-7B-128K, GLM-4-9B-1M, and Llama-3-8B-1M in Figure 5.

K Efficiency Evaluation

Table 9 summarizes the measured batch capacities and relative decode throughput from the latest runs reported in Table 1(a). The vLLM speedup compares each method at its own maximum supported batch, rather than at matched concurrency. The HF column reports the measured capacity of the separate Hugging Face Transformers implementation. Decode throughput excludes prefill and should not be interpreted as end-to-end serving throughput.

Single-stream latency and capacity-level throughput answer different questions: a method can serve more concurrent requests without reducing latency for an individual stream. Table 10 retains the

Table 9: Measured capacity and relative decode throughput, Qwen3-8B on QA1, budget $b = 4{,}096$, $\gamma = 1.5$, 64 outliers, one 80 GB accelerator. Speedup is vLLM decode throughput relative to full attention, each at its own capacity.

Context	Maximum measured batch $B_{\max}$			vLLM speedup
(tokens)	Full vLLM	Ours vLLM	Ours HF	($\times$)
16,384	16	20	90	1.24
32,768	8	20	90	2.12
65,536	4	20	84	3.48
126,976	2	16	72	6.01

Table 10: Single-batch prefill and decode latency at different context lengths. Full attention is the reference baseline with sufficient resources. The "w. CPU" rows denote evaluation with value cache offloading to host memory.

Model	Dataset	Context Length	Prefill Latency (s)			Decode Latency (s)		
			Full Attn.	ShadowKV	PrescienceKV (Ours)	Full Attn.	ShadowKV	PrescienceKV
Llama-3.1-8B	HotpotQA	8192	0.016	3.84	**0.027**	0.886	4.13	**1.32**
Llama-3-8B-1M	QA1	65536	0.025	15.05	**1.89**	10.20	16.01	**15.23**
Llama-3-8B-1M	QA1	131072	25.61	39.96	**26.81**	33.68	41.20	**39.66**
Llama-3-8B-1M w. CPU	QA1	65536	13.55	19.59	**13.99**	41.06	26.72	**22.66**
Llama-3-8B-1M w. CPU	QA1	131072	40.72	47.15	**41.55**	87.65	58.02	**55.80**

separate single-batch prefill and decode latency comparison for full attention, ShadowKV, and PrescienceKV.

Figure 6 visualizes the latest relative throughput and measured capacities from Table 9.

L Memory Evaluation

We report the vLLM KV-pool size, auxiliary state, and host-resident value storage separately in Table 11, Table 12, and Table 13. The rows use different batch sizes and therefore describe memory configurations at different concurrency levels, rather than a matched-batch comparison of total memory consumption.

The sparse-attention budget specifies how many entries are selected, not how many bytes remain resident on the device. The KV-pool column reports the capacity vLLM makes available after model weights and other allocations; it is not a measurement of live KV tensors alone. Auxiliary state is reported in MiB per layer and must be accounted for across all layers, including any retained buffers not already counted elsewhere. Host-resident storage is reported separately and does not constitute a reduction in total system memory.

A resident-memory comparison must count each allocation once and use the same batch size, context length, and measurement phase for both methods. For a model with L layers, a per-layer state allocation of s MiB contributes $Ls/1024$ GiB to device memory. Thus, the KV-pool size alone does not establish the total device-memory saving; peak prefill memory and steady-state decode memory must also be distinguished.

M Ablations

We provide an ablation on the refresh schedule of the frozen historical KV subset within a turn. The default policy uses the original linear surrogate and windowed attention with plain top-k selection at step 0 of each turn, then freezes the final historical subset and appends all newly generated KV entries. The step-0 window contains the first decode query and the last 32 prompt queries, or all available prompt queries for shorter prompts. In Table 14, we compare this policy with additional within-turn reconstructions and observe unchanged accuracy point estimates for the tested schedules. In Table 15, we vary the prefilter count per branch and query head before union. Increasing this count

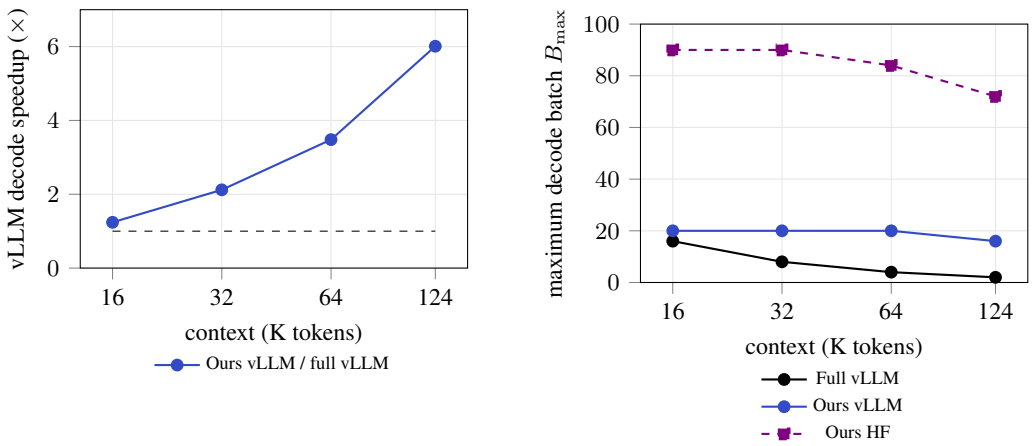

Figure 6: Deployment scaling for Qwen3-8B with historical budget $b = 4{,}096$, $\gamma = 1.5$, and 64 outliers on one 80 GB accelerator. Left: vLLM decode-throughput speedup at each method's own maximum batch. Right: measured batch capacities of the separate vLLM and HF implementations.

Table 11: Memory configuration at 32,768 tokens for Qwen3-8B in vLLM, with a sparse-attention budget of 4,096 entries. KV pool denotes vLLM-reported HBM capacity; state denotes per-layer auxiliary storage; host denotes CPU-resident value storage. Batch sizes differ across rows. These components are not a total-memory reduction metric.

Method	Batch	KV pool (GiB)	State (MiB/layer)	Host (GiB)
Full attention (HBM)	8	51.13	–	–
Ours (HBM)	20	25.40	730.0	–
Full attention (CPU)	16	48.60	72.0	1.4
Ours (CPU)	24	20.34	876.0	0.6

changes the candidate coverage, with an accuracy improvement for Qwen3-8B at the largest tested factor. The factor scales b_c relative to b, not the size of the union $\mathcal{C}$. Thus, factors below one do not imply that fewer than b candidates remain after union across branches and heads. The default factor is $\gamma = 1.5$, and all accuracy evaluations are tested with a batch size of $B = 1$.

M.1 Prompt-Query Window Size

We conduct an additional evaluation on varying number of w_q, while always including the first decode query. Table 16 reports GLM-4-9B-1M at 64K context with $b = 4{,}096$, $\gamma = 1.5$, $o = 64$, and accuracy batch size $B = 1$. The prompt-query count w_q is distinct from the recent historical candidate count $w_{\text{rec}} = 128$.

The default window increases the mean from 84.4 to 85.6, an improvement of about 1.2 percentage points over current-query-only selection. The improvement is modest in this fixed-request long-context setting.

Table 12: Memory configuration at 65,536 tokens for Qwen3-8B in vLLM, with a sparse-attention budget of 4,096 entries. Column definitions follow Table 11; batch sizes differ across rows.

Method	Batch	KV pool (GiB)	State (MiB/layer)	Host (GiB)
Full attention (HBM)	4	51.14	–	–
Ours (HBM)	20	22.59	810.0	–
Full attention (CPU)	8	48.74	68.0	1.4
Ours (CPU)	24	16.96	972.0	0.5

Table 13: Memory configuration at 126,976 tokens for Qwen3-8B in vLLM, with a sparse-attention budget of 4,096 entries. Column definitions follow Table 11; batch sizes differ across rows.

Method	Batch	KV pool (GiB)	State (MiB/layer)	Host (GiB)
Full attention (HBM)	2	51.14	–	–
Ours (HBM)	16	24.13	768.0	–
Full attention (CPU)	4	48.89	64.0	1.4
Ours (CPU)	20	17.38	960.0	0.5

Table 14: Reconstruction schedules for the final historical KV subset, with step 0 denoting the first decode step of a turn. Both selection stages run at reconstruction. Between reconstructions, attention uses the retained subset and all appended entries without further top-k selection.

Selection refresh	Llama-3-8B-1M		Qwen3-8B	
	acc	vs. base	acc	vs. base
step 0 only (used)	0.667	(reference)	0.333	(reference)
steps 0, 16	0.667	$+0.000 \pm 0.000$	0.333	$+0.000 \pm 0.068$
steps 0, 4, 16, 64	0.667	$+0.000 \pm 0.000$	0.333	$+0.000 \pm 0.068$

Table 15: Prefilter count per score branch and query head as a multiple of the final historical budget (`grad_factor`). The union size $D = |\mathcal{C}|$ depends on overlap between selected indices and is not the factor times b.

Prefilter factor	Llama-3-8B-1M		Qwen3-8B	
	acc	vs. base	acc	vs. base
$0.5\times$	0.667	(reference)	0.333	(reference)
$0.25\times$	0.633	-0.033 ± 0.058	0.300	-0.033 ± 0.076
$1.0\times$	0.667	$+0.000 \pm 0.000$	0.333	$+0.000 \pm 0.068$
$2.0\times$	0.667	$+0.000 \pm 0.048$	0.400	$+0.067 \pm 0.082$

Table 16: Prompt-query window ablation on eight RULER tasks with GLM-4-9B-1M at 64K. Each window contains w_q prompt queries plus the first decode query, so $w_q = 0$ uses only the current query.

Prompt queries w_q	MQ	MV	MK1	S1	VT	FWE	QA-1	QA-2	Mean
0	92.5	95.8	100.0	100.0	100.0	66.7	63.3	56.7	84.4
8	95.0	97.5	96.7	100.0	100.0	67.8	63.3	60.0	85.0
32 (default)	95.8	97.5	100.0	100.0	100.0	67.8	63.3	60.0	**85.6**
Full attention	100.0	95.8	100.0	100.0	100.0	73.3	66.7	60.0	87.0