HORIZONKV: FUTURE-AWARE KVCACHE REUSE VIA LIGHTWEIGHT RELEVANCE ESTIMATION FOR INTERLEAVED AGENT CONTEXTS

Anonymous authors
Paper under double-blind review

ABSTRACT

In agent scenarios, continuously interleaved historical and new context makes KVCache reuse essential for reducing prefill cost, while selective recomputation is needed to preserve accuracy. However, existing methods select recomputation positions based on fixed positions or historical relevance, which can miss important positions in dynamically evolving agent contexts. We observe that the next token query helps identify historical positions relevant to subsequent generation. This motivates future-aware KVCache reuse for interleaved contexts, which introduces two challenges. First, this query is unavailable when recomputation positions are selected, and obtaining it requires full prefill. Second, selected positions still attend to the full history during recovery, although most attention falls on a small subset of keys.

To address these challenges, we propose *HorizonKV*, a two stage sparse recovery framework guided by **future relevance estimated** from current dynamic content. First, we propose **future relevance guided recomputation selection**, which averages shallow layer queries at dynamic positions to score historical positions and select them for recomputation within the budget. Second, we propose **future relevance guided shared key recovery**, which reuses these scores to construct a shared KV set, reducing recovery computation without additional dense attention scoring. On agent traces, HorizonKV improves the average score by up to 64.9% over CacheBlend under the same 10% recomputation ratio. On HotpotQA agent trace, it achieves $5.85\times$ TTFT speedup over CacheBlend at comparable accuracy.

1 INTRODUCTION

Large language models (LLMs) have evolved from single-turn QA systems (Liu et al., 2024) into autonomous agents performing long-horizon tasks through iterative interactions, tool use, and multi-agent collaboration (Wang et al., 2024; Tran et al., 2025; Zeng et al., 2024; Ge et al., 2023). Coding and research agents, including Codex(OpenAI, 2026), Claude Code(Anthropic, 2026), and SWE-agent (Yang et al., 2024), exemplify this trend through planning, retrieval, and reasoning workflows. Unlike isolated chatbot requests, agent interactions accumulate user instructions, tool calls, retrieved information, and intermediate responses (Wu et al., 2024). These interleaved agent contexts mix reusable history with new information (Figure 1).

Growing interleaved contexts incur substantial prefill overhead as requests repeatedly process accumulated histories (Kwon et al., 2023; Zheng et al., 2024) (Figure 1(b)). KVCache reuse reduces this cost (Figure 2(a)), but recomposing cached segments can change their preceding context and make cached states stale. Selectively recomputing important positions recovers accuracy while maintaining efficiency (Hu et al., 2024; Yao et al., 2024).

Existing KVCache reuse methods select recomputation positions using predefined structures or historical relevance signals (Figure 2(b)). Fixed locations or cache layouts cannot adapt to changing context relevance. Historical KVCache relevance provides stronger task awareness, but relies on past reuse behaviors rather than future attention demands in evolving agent contexts.

We observe that the next token query provides a strong signal for identifying historical positions relevant to subsequent generation. As shown in Figure 3(b), ranking cached positions with this

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.

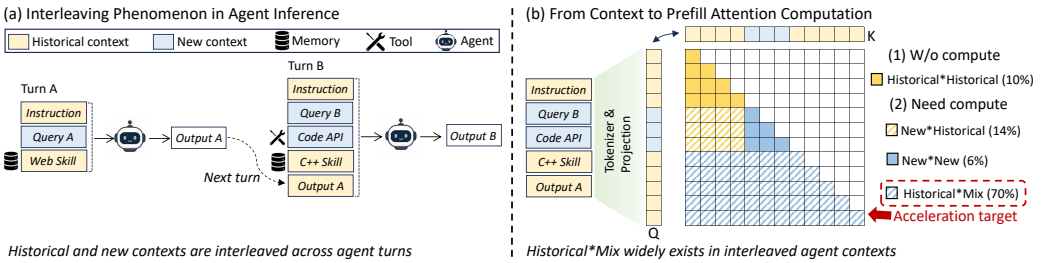


Figure 1: Interleaved agent contexts require recovering historical KVCache affected by new content.

query achieves substantially better alignment with output-relevant recomputation than existing static or historical signals. Furthermore, our analysis reveals that KVCache recovery exhibits dual-axis sparsity: only a small subset of cached positions requires recomputation, and these positions depend on a compact subset of historical keys. These observations motivate efficient KVCache reuse by progressively reducing recovery from full $Q \times K$ computation to sparse query and key dependencies.

Exploiting this dual-axis sparsity poses two challenges. First, the next token query is unavailable at reuse-decision time; obtaining it requires full prefill, eliminating the benefit of KVCache reuse. Second, key dependencies vary across selected recomputation positions, making efficient recovery of the required historical keys challenging without expensive dense attention computation.

To address these challenges, we propose **HorizonKV**, a two stage sparse recovery framework guided by future relevance estimated from dynamic content. First, *future relevance guided recomputation selection* uses averaged shallow layer queries at dynamic positions to score historical positions and select Q_s for recomputation within the budget. Second, *future relevance guided shared key recovery* reuses these scores to construct a shared key set K_s without additional dense attention scoring. Together, the two stages reduce recovery from $Q \times K$ to $Q_s \times K$ and finally to $Q_s \times K_s$.

Our contributions are summarized as follows:

- We propose future relevance guided recomputation selection, using shallow layer queries to estimate relevance and select important positions under limited budgets.
- We propose future relevance guided shared key recovery, reusing relevance scores to select shared historical keys with low overhead.
- We conduct extensive experiments on interleaved agent traces across seven LongBench (Bai et al., 2023) workloads, showing accuracy gains (Figure 2(d)). HorizonKV improves the average score by up to 64.9% over CacheBlend under a 10% recomputation budget. On HotpotQA, it achieves a $5.85\times$ TTFT speedup over CacheBlend at comparable accuracy.

2 Related Work

Existing KV cache reuse methods can be categorized by their recomputation selection signals: non-attention signals and historical relevance signals. HorizonKV instead uses estimated future relevance to guide selection.

Fixed-position selection. These methods determine reuse and recomputation positions using predefined structures or positional heuristics rather than runtime attention signals. Prefix caching systems, including vLLM's PagedAttention (Kwon et al., 2023) and SGLang (Zheng et al., 2024), reuse KV states across identical prefixes. PromptCache (Gim et al., 2023) extends reuse to non-prefix contexts. EPIC (Hu et al., 2024) introduces position-independent context caching and exploits static attention patterns to reduce recomputation overhead, whereas KVLink (Yang et al., 2026) enables independent document KV caching through position adjustment and attention restoration. These predefined structures and static importance patterns limit adaptability to evolving agent contexts.

Historical relevance selection. These methods use historical relevance to select recomputation positions. CacheBlend (Yao et al., 2024) combines layer-wise deviations between cached and recomputed KV states with adjacent-layer correlations to identify and selectively recompute high-

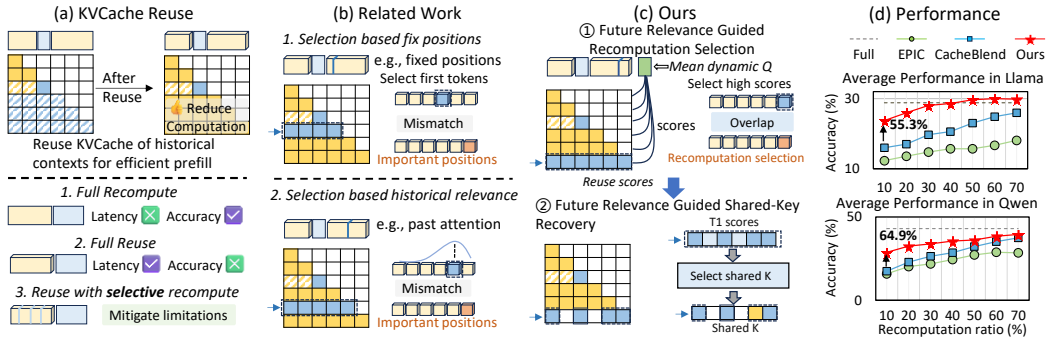


Figure 2: Estimated future relevance guides recomputation selection and shared key recovery.

deviation tokens. These signals improve task awareness over structural strategies but may not reflect future attention requirements as agent contexts evolve.

SpecCache (Wen et al., 2026) uses deep-layer hidden-state norms from a lightweight speculative model to select recomputation positions in RAG. Lookahead Q-Cache (Wang et al., 2025) and Judge Q (Liu et al., 2026) use pseudo and trainable queries, respectively, to guide KV eviction rather than reuse. HorizonKV estimates future relevance from current dynamic content within the target model to guide both recomputation selection and shared key recovery.

Recent studies have also explored KV sharing in multi-agent scenarios. RelayCaching (Geng et al., 2026) primarily reuses KV states associated with upstream agents' outputs, reducing redundant computation when downstream agents incorporate those outputs. KVCOMM (Ye et al., 2026) reduces redundant KV computation or communication through cross-agent KV reuse. Unlike these efforts focused on sharing and transferring KV states associated with agent outputs, HorizonKV focuses on fine-grained recomputation position selection for dynamically evolving, interleaved agent contexts (Figure 2(c)).

3 Observations

KVCache reuse reduces prefill computation, but evolving contexts create mismatches between reused and ideal KVCache(Wen et al., 2026). We investigate which cached positions need recomputation and which historical dependencies their recovery requires.

On representative interleaved agent traces, Llama-3.1-8B-Instruct(Grattafiori et al., 2024) reveals future-relevant KVCache recovery sparsity along both query and key dimensions.

Static structures and historical relevance signals used by existing selectors may not reflect future output demands. We quantify each cached block's contribution through its causal gain:

$$g_b = D_{\mathrm{KL}}(p_{\mathrm{full}} \| p_{\mathrm{reuse}}) - D_{\mathrm{KL}}(p_{\mathrm{full}} \| p_b), \tag{1}$$

where p_b denotes the output distribution after recomputing block b. A larger g_b indicates a larger contribution to recovering Full-Prefill behavior.

At the same recomputation budget, existing selectors cover a limited fraction of important positions (Figure 3(a)). Ranking with the next token query ("Future Q" in the figure) substantially improves this coverage (Figure 3(b)). Here, the next token query is the query of the first response token processed after full prefill; obtaining it requires completing prefill and processing that token.

Observation 1. *Future attention queries better capture future output demands, enabling more accurate recomputation position selection.*

Efficient recovery requires recomputation demand to concentrate on few cached positions. We measure future-relevant state discrepancy as

$$u_t = \frac{a_t^* d_t}{\sum_j a_j^* d_j}, \tag{2}$$

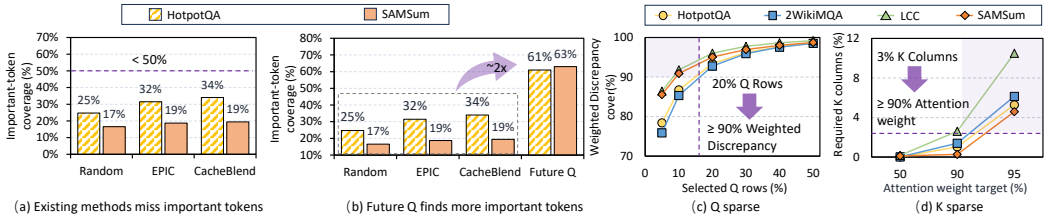


Figure 3: Future queries improve selection (a,b); recovery is sparse along Q and K (c,d).

where d_t is the mean squared V discrepancy between reused and Full-Prefill states, normalized over cached positions per layer and then averaged across sampled layers. Here, a_t^* is the head-averaged next token attention weight, and j ranges over cached positions.

Figure 3(c) shows that a small fraction of token rows covers most of this weighted discrepancy across workloads.

Observation 2. *Future-relevant state discrepancy concentrates on a small subset of cached positions, motivating sparse recovery along the query dimension.*

After selecting query positions, recovery requires identifying their historical key dependencies. We measure the minimum number of keys needed to preserve a target attention mass:

$$S_\tau = \min \left\{ s : \sum_{j \in \text{Top-}s(a)} a_j \geq \tau \right\}. \tag{3}$$

Figure 3(d) shows that most attention mass is concentrated on a small subset of cached keys. This motivates restricting the key positions read during recovery.

Observation 3. *Attention from selected recomputation positions is concentrated on a small subset of historical keys, motivating key-sparse recovery.*

Together, future relevance and concentrated key dependencies motivate dual-axis recovery: HorizonKV selects few query positions and their shared historical keys, reducing $Q \times K$ to $Q_s \times K$ and then $Q_s \times K_s$, where $|Q_s| \ll |Q|$ and $|K_s| \ll |K|$.

4 HorizonKV

HorizonKV estimates future relevance—the relevance of historical positions to subsequent generation—from current dynamic content. Technique 1 (T1) uses these scores for recomputation selection; Technique 2 (T2) reuses them for shared key recovery. Together, they reduce attention computation along both query and key dimensions (Figure 4).

4.1 Problem Formulation

Let $x_{1:T}$ denote the current input, with reusable cached positions $\mathcal{C}$ and newly inserted positions $\mathcal{D}$. Cached positions retain layer-wise states $(K_{\mathcal{C}}^{(\ell)}, V_{\mathcal{C}}^{(\ell)})$ from previous contexts. Recomposing cached segments with new user messages, tool results, and peer responses can change their preceding context, motivating selective recomputation for accurate reuse.

HorizonKV selects a set of recomputation positions $\mathcal{I}_Q$ and a set of key positions $\mathcal{I}_K$. At each recovery layer, the selected queries $Q_s = Q[\mathcal{I}_Q]$ attend to $(K_s, V_s) = (K[\mathcal{I}_K], V[\mathcal{I}_K])$, giving the sequence

$$Q \times K \longrightarrow Q_s \times K \longrightarrow Q_s \times K_s.$$

The recomputation budget controls which states are updated, while the key budget controls which states are read during recovery.

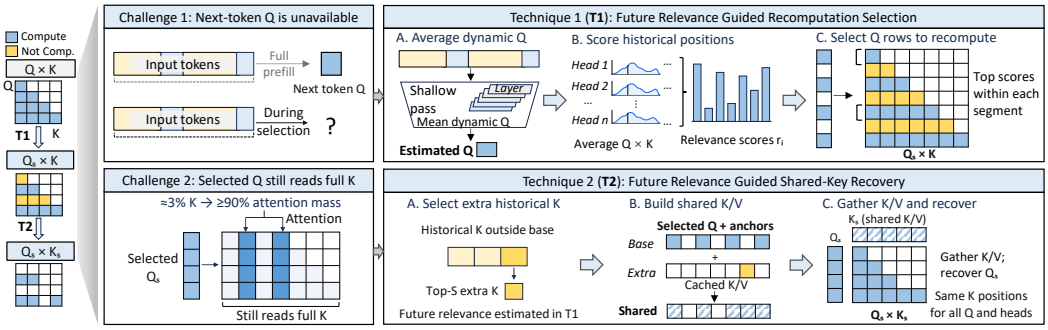


Figure 4: HorizonKV estimates future relevance to select Q rows (T1), then reuses the scores to select shared keys (T2).

4.2 T1: Future Relevance Guided Recomputation Selection

Motivation. We estimate future relevance before the next token query is available. Dynamic content provides a natural starting point: a new question specifies what to retrieve from the history, while tool results and peer responses can change which historical information is needed next. The queries at these positions incorporate their visible context and already provide the model's representation for scoring keys. On Llama-3.1-8B-Instruct, averaging these shallow queries yields higher Top-k recall of historical positions selected by the next token query than using the last dynamic query or a random query across the evaluated per-segment budgets (Figure 5(b)). Here, the next token query is the query of the first response token at the same selector layer, following full prefill. This finding motivates using the dynamic-query average to select historical positions for recomputation.

Future relevance estimation. HorizonKV runs the recomposed input up to a shallow selector layer ℓ_s, then averages the queries at dynamic positions:

$$\widehat{q}_h = \frac{1}{|\mathcal{D}|} \sum_{t \in \mathcal{D}} Q_{t,h}^{(\ell_s)}. \tag{4}$$

The average combines information distributed across dynamic tokens into one average query per head. It reduces the query sequence length used for selection to one, while keeping the head dimension unchanged. If no dynamic position is available, the final prompt query is used.

For a cached position i, we compute its relevance score using a calibrated head set $\mathcal{H}_{\text{cal}}$:

$$e_{h,i} = \frac{\widehat{q}_h^{\top} K_{i,g(h)}^{(\ell_s)}}{\sqrt{d}}, \qquad h \in \mathcal{H}_{\text{cal}}, \tag{5}$$

$$r_i = \sum_{h \in \mathcal{H}_{\text{cal}}} w_h e_{h,i}, \tag{6}$$

where d is the head dimension and $g(h)$ maps a query head to its KV head. Fixed nonnegative weights w_h incorporate head calibration and KV-head grouping (Appendix A.2). By linearity, scoring a key with the averaged query equals averaging the dynamic queries' dot-product scores for that key. The resulting r_i estimates each historical position's relevance to subsequent generation.

Segment-aware Q selection. Reusable segments differ in length and content. HorizonKV assigns a recomputation budget R_s to each segment s, then selects its highest-scoring positions. Let $\mathcal{J}_s$ denote the positions scored by the selector within segment s, and $\mathcal{A}_s$ its mandatory sink and tail positions. The selected set is

$$\mathcal{I}_Q = \mathcal{D} \cup \bigcup_s \left[\mathcal{A}_s \cup \text{Top}_{R_s}\left(\{r_i : i \in \mathcal{J}_s\}\right) \right], \tag{7}$$

where Top_{R_s} returns position indices. The union counts each position once, including any scored position that is also mandatory. Segment budgets and sink/tail rules are specified in Appendix A.3. Dynamic positions are recomputed along with selected cached positions; other cached positions reuse their stored states. Subsequent recovery therefore operates on $Q_s \times K$.

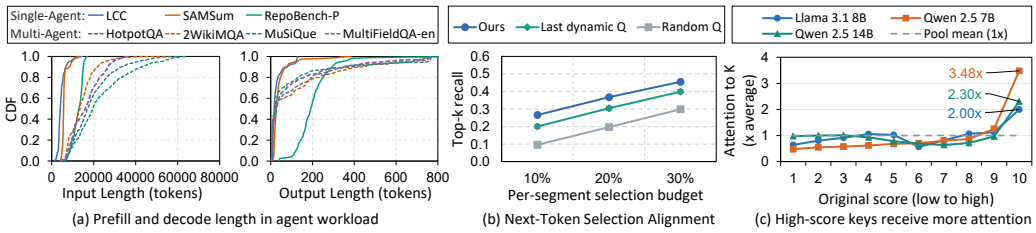


Figure 5: (a) Prefill/decode lengths. (b) Next token top-k recall. (c) Selected-query attention per key across score bins, relative to the scored-pool mean. See Appendix C.2 for diagnostic details.

4.3 T2: Future Relevance Guided Shared Key Recovery

Motivation. After Q selection, the selected queries still attend to the full history. Selecting keys directly from their attention weights would require computing the full attention scores $Q_s K^\top$, incurring part of the cost that key sparsity is intended to save. We find that, among scored historical positions, those with the highest T1 relevance scores receive above-average attention from the selected queries. In diagnostics on three models, the highest-scoring 10% of scored historical positions receive 2.00–$3.48\times$ the pool-average attention per key at a 10% Q budget, even after excluding attention to each query's own position (Figure 5(c)). This observation identifies useful information in the existing relevance scores. We reuse them as a low-cost heuristic for constructing a shared key set, with the resulting recovery evaluated in Section 5.4.

Shared K selection. HorizonKV first retains a base key set $\mathcal{B} = \mathcal{I}_Q \cup \mathcal{A}$, where $\mathcal{A} = \bigcup_s \mathcal{A}_s$. Historical positions outside this set can still supply useful context through their cached K/V. We therefore allocate an additional budget S to these positions. To keep this selection inexpensive, we rank the remaining candidates using the existing relevance scores r_i:

$$\mathcal{I}_{\text{extra}} = \text{Top}_S\big(\{r_i : i \in \mathcal{C} \setminus \mathcal{B}\}\big), \tag{8}$$

$$\mathcal{I}_K = \mathcal{B} \cup \mathcal{I}_{\text{extra}}. \tag{9}$$

T1 selects recomputation positions within each segment; T2 allocates additional keys globally across remaining cached positions. These additional positions contribute their cached K/V without increasing the number of recomputed queries. The budget S specifies the number of additional keys, up to the available candidates; the complete key set also includes $\mathcal{B}$. The T1/T2 ablation evaluates this complete shared key design; Appendix B compares its shared layout with per-head recovery.

Shared K/V recovery. All selected queries and attention heads use the same key positions, while each KV head retains its own key and value vectors. HorizonKV gathers these positions into compact tensors:

$$K_s = \text{Gather}(K, \mathcal{I}_K), \qquad V_s = \text{Gather}(V, \mathcal{I}_K). \tag{10}$$

At each recovery layer, the selected queries attend to the gathered K/V:

$$O_s = \text{Softmax}\left(\frac{Q_s K_s^\top}{\sqrt{d}} + M_{\text{causal}}\right) V_s. \tag{11}$$

K/V at recomputed positions are updated with the current layer's states; other retained positions supply cached states. The causal mask uses positions in the recomposed input, so each query can read only preceding positions and itself. Gathering preserves these positions and their positional encodings. The recovered rows continue through subsequent layers, with unselected cached positions retaining their stored K/V. Using a shared position set gives attention a compact key dimension across queries and heads, reducing recovery to $Q_s \times K_s$.

4.4 Computational Cost

Let $R = |\mathcal{I}_Q|$, $M = |\mathcal{I}_K|$, and H be the number of query heads. Per recovery layer, dense attention costs $\mathcal{O}(T^2 H d)$; T1 reduces it to $\mathcal{O}(RTHd)$, and T2 further reduces it to $\mathcal{O}(RMHd)$. Both stages use the same shallow pass and relevance scores. After query pooling, relevance scoring costs $\mathcal{O}(|\mathcal{C}|\,|\mathcal{H}_{\text{cal}}|d)$. T2 adds global key selection and K/V gathering, with no additional model forward pass. End-to-end latency includes the shallow pass, selection, gathering, and recovery.

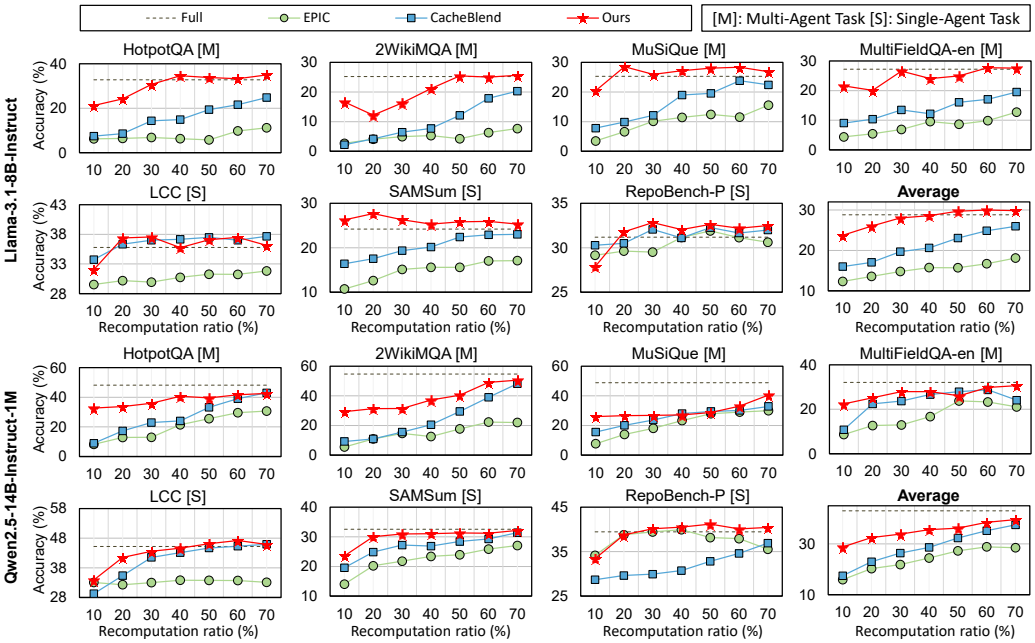


Figure 6: LongBench score versus Q budget. [M]/[S] denote multi-/single-agent traces.

5 Evaluation

5.1 Evaluation Setup

Agent Configuration. We primarily use Llama-3.1-8B-Instruct (Grattafiori et al., 2024), deployed on 8 NVIDIA A800 80GB GPUs(NVIDIA, 2026). We additionally evaluate accuracy on Qwen2.5-14B-Instruct-1M (Yang et al., 2025) (Figure 6). We use AutoGen (Wu et al., 2024) to construct solver and judge workflows, and use vLLM (Kwon et al., 2023) for long-context inference and KVCache management.

Datasets and Metrics. We evaluate HorizonKV on seven LongBench datasets (Bai et al., 2023) covering four task categories: multi-document QA, single-document QA, summarization, and code completion. Each instance is converted into an agent trace using AutoGen (Figure 5(a)). All methods use the same workload construction, with each method's generated upstream messages passed to downstream agents. Accuracy is measured using the original dataset metrics (e.g., F1, ROUGE-L, and code similarity), while efficiency is evaluated by TTFT and recomputation ratio.

Baselines and Configurations. We compare HorizonKV with EPIC (Hu et al., 2024)[ICML'25], which uses static position selection, and CacheBlend (Yao et al., 2024)[EuroSys'25], which uses historical KVCache relevance for recomputation selection. **Full Recomputation** recomputes all reusable KVCache positions and serves as the accuracy reference. All selective methods use the same nominal historical recomputation budget. Unless stated otherwise, HorizonKV adopts a layer-4 shallow proxy and an additional key budget of $S = 1000$ beyond the base set $\mathcal{B}$.

Implementation Details. We consider two agent topologies: **Single-Agent topology**, where one agent processes each task with interleaved context segments, and **Multi-Agent topology**, where multiple agents (`solver_a` $\rightarrow$ `solver_b` $\rightarrow$ `judge`) collaborate through multi-turn interactions. Both settings use interleaved workloads with heterogeneous reusable and dynamic segments.

5.2 Accuracy Evaluation

We first compare HorizonKV's task scores with existing reuse methods under the same Q-recomputation budget, using Full Prefill as a reference.

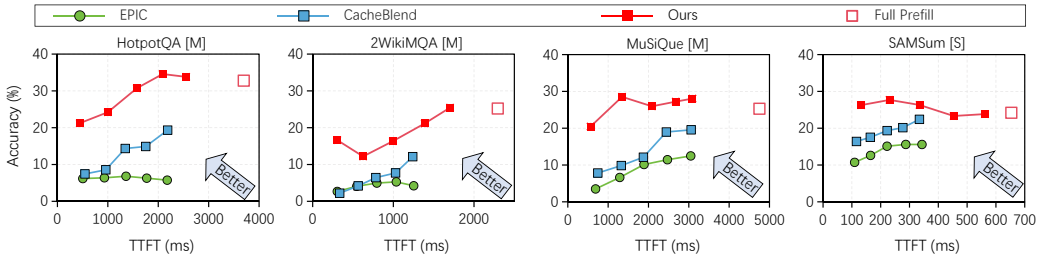


Figure 7: LongBench score–TTFT trade-off on representative workloads.

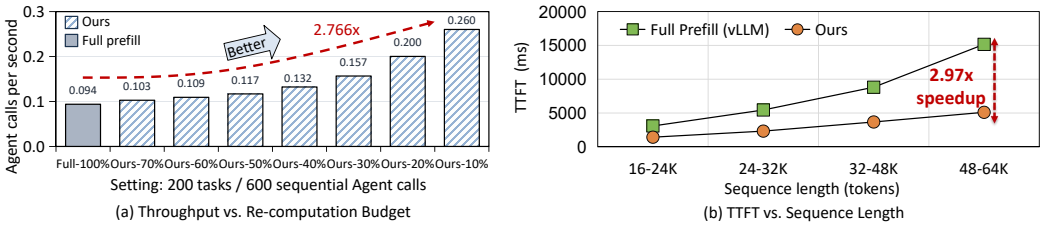


Figure 8: (a) HotpotQA sequential throughput (200 tasks, 600 agent calls); percentages denote Q-recomputation budgets. (b) TTFT versus sequence length.

Overall trend. Figure 6 compares all methods under Q-recomputation budgets from 10% to 70%. On Llama-3.1-8B-Instruct, HorizonKV achieves the largest gains under limited budgets, improving the average score by 55.3% over CacheBlend at 10% budget and 55.1% at 20%. On Qwen2.5-14B-Instruct-1M, the gain reaches 64.9% at a 10% budget. More recomputation at higher budgets narrows the gap by compensating for imperfect selection.

Multi-agent traces. HorizonKV shows its strongest advantage on multi-agent traces, with larger average gains on multi-hop QA workloads. These traces combine documents and intermediate agent messages, where context changes can alter which historical positions matter for subsequent generation. Future-aware relevance guides recomputation toward these positions.

Single-agent traces. On Llama, HorizonKV also improves average scores on single-agent traces, with smaller gains: 7.2% over CacheBlend at a 10% budget and 14.8% at 20%, averaged over SAMSum, LCC, and RepoBench-P. This is consistent with fewer heterogeneous context changes, where historical relevance remains relatively effective. Notable gains on SAMSum demonstrate the benefits of future-aware recomputation beyond multi-agent scenarios.

5.3 Efficiency Evaluation

We evaluate per-request TTFT, end-to-end agent throughput, and context-length scaling, then break down the remaining runtime after dual-axis sparsification.

Accuracy–TTFT trade off. HorizonKV expands the Pareto frontier on all four workloads (Figure 7). On HotpotQA, it achieves a $5.85\times$ TTFT speedup over CacheBlend at comparable accuracy. On MuSiQue, it achieves a 12.8% higher score than Full Prefill with $3.53\times$ TTFT speedup. These results show that selective recomputation with shared key recovery can retain strong task performance while reducing TTFT. Full Prefill is a reference configuration rather than an upper bound on task score.

Sequential throughput. Figure 8(a) measures completed agent calls, not isolated attention kernels. HorizonKV achieves $1.67\times$–$2.77\times$ Full-Prefill throughput as the Q budget decreases from 30% to 10%. Tighter budgets eliminate more prefill work through dual-axis recovery; at higher budgets, recomputation again dominates the request.

Table 1: Diagnostic GPU timing at a 30% Q budget and $S = 1000$, averaged over the seven workloads. Shares use the reported timer total.

	T1			T2	Total
Metric	Relevance estimation	Q selection	Subtotal	Selection + recovery	Sparse recovery
Time (ms)	141.71	96.35	238.06	902.91	1140.97
Share (%)	12.42	8.44	20.86	79.14	100.00

Table 2: T1 and T2 ablation with $S = 1000$ for T2. TTFT speedups are relative to Full Recomputation ($1.00\times$); the T2 incr. column is relative to T1. Values are macro-averaged over matched tasks at fixed Q budgets.

	LongBench score			TTFT speedup		
Q budget	Random-Q	T1	T1+T2	T1	T1+T2	T2 incr.
10%	20.08	22.13	**23.73**	$4.19\times$	$\mathbf{6.75}\times$	$1.61\times$
30%	23.51	26.34	**28.04**	$1.72\times$	$\mathbf{2.09}\times$	$1.22\times$

Scaling with context length. HorizonKV's advantage grows with context length, reaching $2.97\times$ TTFT speedup in the longest bin (Figure 8(b)). Dense prefill recomputes the entire history, whereas HorizonKV constrains recovery along both query and key axes.

Runtime breakdown. Table 1 reports diagnostic GPU timing at a 30% Q budget. T1's relevance estimation and Q selection account for 20.86% of the reported timer total; T2's shared key selection and recovery account for 79.14%. These shares describe GPU intervals, not an additive TTFT decomposition (Appendix D.2). Recovery dominates the recorded time at this budget.

5.4 Ablation Results

We isolate the contributions of **future relevance guided recomputation selection (T1)** and **future relevance guided shared key recovery (T2)**, which introduce Q- and K-axis sparsity, respectively.

Incremental Accuracy gains. To isolate T1's future relevance signal, Random-Q retains T1's segment structure, sink/tail rules, and recomputation budget, replacing only the average query with a deterministic norm-matched random direction. Table 2 reports the incremental LongBench gains. At 10% and 30% budgets, T1 improves the macro-average score over Random-Q by 10.2% and 12.0%, respectively. Adding T2 at the same Q budgets yields further gains of 7.2% and 6.5%, and overall gains of 18.2% and 19.3% over Random-Q, without dataset-specific selection of S.

Incremental TTFT gains. Table 2 separates Q- and K-axis contributions via macro-average TTFT speedups across workloads over Full Recomputation. At a 10% Q budget, T1 achieves $4.19\times$ speedup, rising to $6.75\times$ with T2; at 30%, these are $1.72\times$ and $2.09\times$. TTFT gains exceed throughput gains (Figure 8) since throughput includes decoding and non-prefill overheads.

6 Conclusion

We presented *HorizonKV*, a future-aware KVCache reuse framework for interleaved agent contexts. Our analysis reveals dual-axis sparsity: few cached positions require recomputation over few historical keys. T1 uses estimated future relevance to select recomputation positions; T2 reuses the scores to select shared historical keys, reducing recovery computation. Experiments on agent traces show up to a 64.9% average score improvement over CacheBlend at a 10% recomputation budget. On HotpotQA, HorizonKV achieves a $5.85\times$ TTFT speedup over CacheBlend at comparable accuracy. Effective KVCache reuse thus requires accurate recomputation position selection and efficient historical dependency selection. Future work could reduce future relevance estimation costs.

AI USE STATEMENT

We used generative AI tools for literature retrieval and discovery and to generate the synthetic agent traces used in our experiments. The authors reviewed and validated all AI-assisted outputs and take responsibility for the final content.

ETHICS STATEMENT

This work focuses on improving the efficiency of large language model inference. We do not introduce new datasets containing personal information or conduct studies involving human subjects. The method may reduce serving cost and memory usage for agent workloads, while inheriting the safety and misuse considerations of the underlying language models.

REPRODUCIBILITY STATEMENT

We describe the algorithm, selector configuration, budget allocation, hardware, software, datasets, and evaluation protocol in the main paper and appendix.

REFERENCES

Anthropic. Claude Code. `https://claude.com/product/claude-code`, 2026. Accessed: 2026-07-28.

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

Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. Openagi: When llm meets domain experts. *arXiv preprint arXiv:2304.04370*, 2023.

Yingsheng Geng, Yuchong Gao, Weihong Wu, Guyue Liu, and Jiang Liu. Relaycaching: Accelerating llm collaboration via decoding kv cache reuse. *arXiv preprint arXiv:2603.13289*, 2026.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *arXiv preprint arXiv:2311.04934*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models. *arXiv preprint arXiv:2410.15332*, 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.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li,

Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Yijun Liu, Yixuan Wang, Yuzhuang Xu, Shiyu Ji, Yang Xu, Qingfu Zhu, and Wanxiang Che. Judge q: Trainable queries for optimized information retention in kv cache eviction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32240–32248, 2026.

NVIDIA. NVIDIA A800 PCIe 80GB. NVIDIA AI Enterprise Documentation, 2026. URL `https://docs.nvidia.com/ai-enterprise/release-4/latest/infra-software/vgpu/reference/ampere.html`. Accessed: 2026-07-28.

OpenAI. Codex. `https://openai.com/codex/`, 2026. Accessed: 2026-07-28.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Wen Jirong. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Yixuan Wang, Shiyu Ji, Yijun Liu, Yuzhuang Xu, Yang Xu, Qingfu Zhu, and Wanxiang Che. Lookahead q-cache: Achieving more consistent kv cache eviction via pseudo query. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34146–34162, 2025.

Zijian Wen, Tao Zhang, Shuangwu Chen, Shenghao Ye, Yu Guo, Qirui Chen, Jingxian Shuai, Yunpeng Hou, Huasen He, et al. Speccache: Speculative kv cache reuse for efficient rag serving. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18861–18871, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. Qwen2.5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. *Advances in Neural Information Processing Systems*, 38:133797–133824, 2026.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:17882–17928, 2026.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

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

SUPPLEMENTARY OVERVIEW

This appendix provides the supporting material referenced in the main text and is organized as follows. Section A gives an end-to-end algorithmic description of HorizonKV, the calibrated head derivation, the segment-aware budget rule, and the gather-with-causal-mask implementation. Section B examines the shared key design in T2 through a direct comparison with per-head recovery in accuracy and TTFT. Section C examines future relevance estimation through a selector-layer sensitivity study. Section D reports a K-support S sweep and diagnostic GPU timings. Section E discusses the relation to cross-agent KV sharing. Section F records trace construction, hyperparameters, sample sizes, hardware, and software.

A. METHOD DETAILS

A.1 END-TO-END HORIZONKV ALGORITHM

Algorithm 1 formalizes the full HorizonKV pipeline. The selector layer ℓ_s produces a shared query–key evidence tensor $e_{h,i}$ that is consumed by both stages, so no additional model forward pass is needed beyond the shallow prefix pass.

Both stages reuse relevance score r_i computed from $e_{h,i}$ in Algorithm 1. T1 applies it to recomputation selection; T2 reuses it for shared key recovery. Their roles differ in row versus column selection and recomputation versus K/V gathering. Appendix B evaluates the shared key design.

A.2 HEAD CALIBRATION

For Llama-3.1-8B-Instruct, we use layer-4 heads $\{12, 14, 17, 22, 27\}$ from the ranking obtained with 50 offline 2WikiMQA attention probes. This set is fixed across all seven workloads. Selected query heads are averaged within each KV-head group, followed by averaging across KV heads. Up to a common scale that leaves rankings unchanged, the corresponding weights in Equation 6 are $\{1/8, 1/8, 1/4, 1/4, 1/4\}$ in the listed head order.

For Qwen2.5-14B-Instruct-1M, we calibrate 40 head weights using 50 attention probes per dataset, pooled into QA, summarization, and code groups. Each head is scored by the overlap between its layer-4 top-30% attention positions and those of the final-layer head average. We subtract the median head score, clip negative values to zero, and normalize the remaining weights. The resulting weights are fixed within each task group across requests and recomputation budgets. Both models use the same dynamic-query pooling and Q/K selection procedures.

Algorithm 1 HorizonKV recovery for one agent call.

Require: Recomposed input $x_{1:T}$; cached positions $\mathcal{C}$; dynamic positions $\mathcal{D}$; selector layer ℓ_s; calibrated heads $\mathcal{H}_{\text{cal}}$ and weights $\{w_h\}$; per-segment budgets $\{R_s\}$; anchor set $\mathcal{A}$; scored segment candidates $\{\mathcal{J}_s\}$; additional key budget S.

1: Run the shallow pass through layer ℓ_s over $x_{1:T}$; obtain $Q^{(\ell_s)}, K^{(\ell_s)}$.
2: $\widehat{q}_h \leftarrow \frac{1}{|\mathcal{D}|}\sum_{j\in\mathcal{D}} Q^{(\ell_s)}_{j,h}$ for $h \in \mathcal{H}_{\text{cal}}$ when $\mathcal{D} \neq \varnothing$ *(average dynamic query)*
3: If $\mathcal{D} = \varnothing$, use the final prompt query for $\widehat{q}_h$.
4: $e_{h,i} \leftarrow \widehat{q}_h^{\top} K^{(\ell_s)}_{i,g(h)}/\sqrt{d}$ for $h \in \mathcal{H}_{\text{cal}}, i \in \mathcal{C}$
5: $r_i \leftarrow \sum_{h\in\mathcal{H}_{\text{cal}}} w_h e_{h,i}$ *(shared relevance score)*
6: $\mathcal{I}_Q \leftarrow \mathcal{D} \cup \bigcup_s \left[\mathcal{A}_s \cup \text{Top}_{R_s}(\{r_i : i \in \mathcal{J}_s\})\right]$ *(T1 index space: rows)*
7: $\mathcal{B} \leftarrow \mathcal{I}_Q \cup \mathcal{A}$
8: $\mathcal{I}_{\text{extra}} \leftarrow \text{Top}_S(\{r_i : i \in \mathcal{C} \setminus \mathcal{B}\})$
9: $\mathcal{I}_K \leftarrow \mathcal{B} \cup \mathcal{I}_{\text{extra}}$ *(T2 index space: columns)*
10: $K_s, V_s \leftarrow \text{Gather}(K, \mathcal{I}_K), \text{Gather}(V, \mathcal{I}_K)$ *(physical layout)*
11: Resume layers $\ell_s{+}1, \ldots, L$ with rows $\mathcal{I}_Q$ attending to columns $\mathcal{I}_K$; positions in $\mathcal{C} \setminus \mathcal{I}_Q$ reuse cached (K, V).

A.3 Segment-Aware Budget Allocation

At nominal recomputation ratio ρ, each reusable segment s receives $R_s = \max(\lfloor \rho \cdot |\mathcal{C}_s| \rfloor,\ R_{\min})$ with a floor $R_{\min}=2$ tokens per segment. Anchors $\mathcal{A}_s$ always include the first two sink tokens and the last position of each segment (skipped when a segment is shorter than four tokens). T1 selects from the scored candidate set $\mathcal{J}_s$ and takes its union with these anchors, as in Equation 7; overlapping positions are counted once. The final count includes per-segment rounding and mandatory positions.

A.4 Physical Gather with Causal Mask

This subsection details the physical gather operation in T2 (Section 4.3). After constructing the shared dependency set $\mathcal{I}_K$, T2 gathers the corresponding KV states into compact tensors K_s and V_s, reducing the key dimension used for recovery from T to $|\mathcal{I}_K|$ while preserving positional and causal semantics on the retained support. Although KV states are stored in a compact layout, each retained entry keeps its pre-gather position metadata $\text{pos}(s) \in \{0, \ldots, T-1\}$. T2 does not reapply RoPE or assign new positions after gathering. Here, the original position refers to the pre-gather canonical position rather than the compact storage index in K_s or V_s. Accordingly, the causal mask between a refreshed query $q_j \in Q_s$ and a gathered key $k_i \in K_s$ is determined by $\text{pos}(k_i) \leq \text{pos}(q_j)$ instead of their compact indices. Refreshed queries obtain positions from $\mathcal{I}_Q$ and apply RoPE consistently. The attention softmax is normalized over the retained, causally visible keys.

A.5 Complexity of the Shallow Selector

The shallow prefill runs through selector layer ℓ_s on the full recomposed context. Its cost depends on the input length and selector depth and is fixed with respect to the recomputation budget ρ, so its relative share is largest when ρ is small. Appendix D.2 reports per-request measurements.

B. Shared Key Recovery

This appendix expands the shared key design in T2. Its purpose is to compare the default shared support against a natural per-head alternative and quantify the resulting accuracy–latency trade-off.

B.1 Per-Head Independent Support

Without sharing, each attention head h would maintain its own key support

$$\mathcal{I}_{K,h} \;=\; \mathcal{B} \cup \text{Top}_S\left(\{e_{h,i}\}_{i\in\mathcal{C}\setminus\mathcal{B}}\right), \tag{12}$$

producing $|\mathcal{H}|$ independently selected index sets that may overlap. Physical gathering then requires head-specific layouts or gathers, potentially duplicating reads of shared positions and increasing layout overhead relative to a common support.

B.2 Per-Head vs Shared: Accuracy and TTFT

We next compare shared-K recovery (Section 4.3) against a per-head gather variant that maintains an independent Top-S support per head at a fixed 30% Q-budget (Table 3). A padded-SDPA implementation of the per-head variant only fits on our 80 GB GPU for the shortest workloads and is 3–5$\times$ slower than physical per-head gather, so we omit it from the table. Per-head gather improves scores on some long-QA workloads but is consistently 1.7–$2.3\times$ slower.

Table 3: Shared-K recovery (HorizonKV default) vs a per-head gather alternative at 30% Q-budget and S=1000. Score / TTFT (ms), 200 tasks per cell (150 for MultiFieldQA).

Workload	Shared-K (default)	Per-head gather
HotpotQA	30.73 / 1576	33.75 / 3423
2WikiMQA	16.34 / 994	23.00 / 2023
MuSiQue	25.95 / 2086	27.40 / 4691
MultiFieldQA	26.57 / 1455	22.75 / 3104
LCC	37.56 / 295	37.18 / 490
SAMSum	26.31 / 335	23.72 / 574
RepoBench-P	32.83 / 528	32.38 / 923

B.3 Interpretation

T1 and T2 reuse the same shallow evidence but map it into different index spaces and different execution operators: T1 selects rows to recompute in the full model forward, whereas T2 selects columns for a compact physical gather. Table 3 directly shows the trade-off: the per-head alternative improves accuracy on some long-QA workloads, whereas the shared support is consistently faster and remains competitive on the seven evaluated workloads. We therefore use the shared support as the latency-oriented default and retain the per-head variant as an accuracy-oriented option.

C. Future Relevance Estimation

T1 uses the average dynamic query at layer ℓ_s to estimate future relevance. This appendix reports selector-layer sensitivity and the protocols for the relevance diagnostics in Figure 5(b,c).

C.1 Selector-Layer Sensitivity

We evaluate selector layer ℓ_s by rerunning the full replay pipeline with $\ell_s \in \{1, 2, 4, 8, 16\}$ under a 30% Q-budget and S=1000. Table 4 reports two representative workloads.

Table 4: Selector-layer sensitivity at 30% Q-budget and S=1000. LongBench score, 200 samples per cell. ℓ_s=4 is our default.

Workload	ℓ_s	Score
HotpotQA	1	21.72
	2	28.38
	4	**30.73**
	8	34.22
	16	31.00
SAMSum	1	23.45
	2	24.11
	4	**26.31**
	8	25.36
	16	24.41

ℓ_s=4 achieves the best score on SAMSum. On HotpotQA, ℓ_s=8 obtains the highest score, but ℓ_s=4 remains within 3.5 points of the best result. We adopt ℓ_s=4 as a workload-agnostic default; task-

specific tuning can occasionally recover an additional 1–3.5 points but is not used in any reported main result.

C.2 Relevance Diagnostics

Figure 5(b) compares queries using the same selector layer, calibrated heads, and historical keys. The next token query is obtained by teacher forcing the trace's first response token after full prefill. Forced anchors are excluded from Top-k recall, and eight random query trials are averaged per task.

Figure 5(c) uses Llama-3.1-8B-Instruct, Qwen2.5-7B-Instruct-1M, and Qwen2.5-14B-Instruct-1M with five tasks per dataset and model. Using a T1 reference with all keys, we average sampled query attention across four layers per task. Score bins cover all scored historical positions, including those retained in the base set. Attention to each query's own position is excluded without renormalization.

C.3 Guidance on Selector-Layer Choice

Following C.1, ℓ_s=4 is the shallowest tested selector within 3.5 points of each workload's best layer-tuned score, balancing selector depth and selection quality without workload-specific layer tuning.

D. Extended Experimental Results

D.1 K-Support S Sensitivity Across Budgets

The main text fixes S=1000. Table 5 further sweeps $S \in \{500, 1000, 2000, 3000\}$ at 10%, 30%, and 50% Q-budgets on two representative workloads.

Table 5: Accuracy / TTFT (ms) as a function of additional key budget S and Q-recomputation budget. 200 samples per cell.

Workload	S	10%	30%	50%
HotpotQA	500	21.71 / 425	30.17 / 1563	34.77 / 2469
	1000	21.23 / 444	30.73 / 1576	33.81 / 2553
	2000	23.38 / 472	30.45 / 1649	35.12 / 2619
	3000	21.17 / 499	30.86 / 1731	34.99 / 2687
SAMSum	500	26.67 / 124	27.16 / 322	23.30 / 547
	1000	26.27 / 132	26.31 / 335	23.84 / 562
	2000	27.12 / 145	23.37 / 363	23.68 / 574
	3000	27.33 / 158	20.81 / 386	23.68 / 576

Increasing S expands the retained key set and raises TTFT, while its effect on task score varies by workload and Q budget. At 30% Q budget, HotpotQA scores range from 30.17 to 30.86, whereas SAMSum scores range from 20.81 to 27.16. These results motivate reporting the full sweep and using the same S=1000 default across workloads.

D.2 Runtime Breakdown Across Budgets

Table 6 reports diagnostic CUDA-event timers for the shallow-forward pass, the selector (top-K over r_i), and the T2 sparse recovery kernel, alongside separately measured TTFT.

The subtotal sums the diagnostic timers, not mutually exclusive components of request latency. Shallow-forward time remains approximately constant across Q budgets, while sparse recovery takes longer as more rows are recomputed. At moderate and high budgets, recovery dominates the reported diagnostic total.

D.3 First-Token Fidelity to Full Prefill

We compare T1 (full K) and T1+T2 against Full Prefill on identical, frozen final-call inputs using Llama-3.1-8B-Instruct. This small diagnostic uses five consecutive tasks per dataset (trace positions

Table 6: Per-request diagnostic timer totals, averaged over all agent calls from 200 tasks. Multi-agent tasks contain three calls per task; single-agent tasks contain one. "Shallow" is the ℓ_s=4 prefix pass; "Selector" is the per-segment top-K; "T2" is the sparse recovery kernel. Diagnostic totals are not a wall-clock decomposition of TTFT. All values in ms.

Workload	Budget	Shallow	Selector	T2	Subtotal	TTFT
HotpotQA	10%	221	225	385	831	447
	30%	223	143	1390	1756	1602
	50%	224	96	2233	2553	2557
MuSiQue	10%	288	272	493	1053	570
	30%	282	172	1800	2254	2063
	50%	288	125	2724	3137	3121
SAMSum	10%	51	78	117	246	137
	30%	51	45	291	387	334
	50%	51	25	514	590	589

51–55), the standard selector, and S=1000. T1 and T1+T2 recompute identical Q positions at each nominal budget. We measure full-vocabulary $D_{\mathrm{KL}}(P_{\mathrm{full}} \| P_{\mathrm{method}})$ in nats for the first output token and count matching top-1 tokens.

Table 7: First-token fidelity to Full Prefill. KL is mean ± sample standard deviation over five tasks; agreement is a count out of five. Both budgets use the same tasks (20 distinct tasks overall).

		KL ↓		Top-1 agreement ↑	
Workload	Q budget	T1	T1+T2	T1	T1+T2
HotpotQA	10%	0.717 ± 0.842	1.378 ± 2.190	3/5	2/5
	30%	0.146 ± 0.105	0.111 ± 0.056	3/5	3/5
2WikiMQA	10%	0.092 ± 0.160	2.854 ± 1.592	5/5	0/5
	30%	0.125 ± 0.204	0.342 ± 0.464	5/5	4/5
LCC	10%	2.102 ± 3.801	2.120 ± 2.035	4/5	3/5
	30%	0.143 ± 0.106	0.174 ± 0.158	5/5	5/5
SAMSum	10%	0.563 ± 0.325	2.413 ± 1.499	3/5	1/5
	30%	0.277 ± 0.258	0.522 ± 0.857	4/5	3/5

For T1+T2, increasing the budget from 10% to 30% lowers mean KL on all four workloads and raises pooled top-1 agreement from 6/20 to 15/20. However, T2 increases mean KL relative to T1 on all four workloads at 10% and three at 30%; fidelity is not uniformly preserved. Repeated Full Prefill runs reproduce the reference on all 20 tasks; full-K gather controls match T1 on eight checked cases. These descriptive results concern only the immediate post-prefill distribution, not full-sequence agreement or task accuracy.

E. Positioning Relative to Adjacent Work

We compare against EPIC and CacheBlend as representative recomputation selection baselines under the same nominal Q budget. Cross-agent KV sharing methods, including RelayCaching and KVCOMM, reuse states across agents to reduce duplicate computation or communication. HorizonKV selects which cached positions to recompute and which historical keys to read within an agent call. These approaches address complementary choices: what KV to share across calls and how to recover reused KV within a call.

F. Reproducibility Details

F.1 AutoGen Trace Construction

We construct two trace topologies from LongBench samples using AutoGen. Multi-agent workloads use a *Solve–Solve–Judge* topology: `solver_a` drafts a candidate answer, `solver_b` reads the

same input together with the first solver's message and proposes a corroboration or correction, and `judge` reads both messages and emits the final answer. Single-agent workloads use one solver call per sample. Each agent uses greedy decoding with per-role `max_tokens` in $\{128, 384, 768, 1024\}$ depending on task type. Consequently, a multi-agent sample contributes three requests to the timing tables, while a single-agent sample contributes one. We refer to these as *AutoGen-constructed traces*.

F.2 HYPERPARAMETERS

Table 8 lists the default configuration. Selector depth, key budget, and anchor rules are shared across workloads; model-specific head configurations are given in Appendix A.2.

Table 8: HorizonKV default configuration; head calibration is detailed in A.2.

Symbol	Value	Description
ℓ_s	4	Shallow selector layer
$\mathcal{H}_{\text{cal}}, w_h$	Model-specific (A.2)	Fixed head aggregation
S	1000	Additional keys beyond the base set
$R_{\min}$	2	Per-segment recompute floor
Anchors $\mathcal{A}$	sink 2 + tail 1	Preserved token positions
Selector kind	`q1_dynamic_seg_dot_sink`	Relevance estimation
Boundary rule	`start`	Segment boundary
Sample size	150–500 tasks	Cohort sizes specified below
Backend attn.	xFormers/SDPA (patched vLLM)	Attention execution backend

F.3 DETERMINISM AND STATISTICAL SIGNIFICANCE

Main accuracy cohorts contain 200 tasks for QA and SAMSum (150 for MultiFieldQA). LCC/RepoBench-P use 500 tasks for Qwen and Llama EPIC/CacheBlend, and 200 for Llama HorizonKV/Full Prefill. Table 2 uses 200 matched tasks per workload (150 for MultiFieldQA).

Our replay pipeline fixes the tasks and prompt structure and uses greedy decoding. Agents run in order, with each method's generated upstream messages replacing the recorded messages in downstream prompts. The segment tokenizer and skill-pool ordering use seed 42. Under a fixed software and model configuration, repeated replays produce identical judge outputs; runtime measurements may still vary with system load. Accuracy is therefore reported once per fixed trace cohort, while TTFT averages all calls in that cohort.

To assess task-sampling variability, we compute a paired Wilcoxon signed-rank test on per-task F1 of complete HorizonKV (T1+T2) vs CacheBlend at 30% Q-recomputation budget on three multi-hop QA workloads, which most directly instantiate the interleaved, evolving-context setting that motivates our design (Table 9). Pairing is exact: the two methods run on the same 200 tasks; we record their judge outputs, score each task with F1, and test the score-delta distribution. We use the two-sided SciPy signed-rank implementation with `zero_method=''wilcox''`; zero-difference pairs are excluded from the rank statistic.

Table 9: Paired Wilcoxon signed-rank test of HorizonKV vs CacheBlend at 30% Q-recomputation budget on three representative multi-hop QA workloads (200 paired agent traces per workload). $\overline{\Delta}$ is the mean per-task F1 gap in percentage points; r is the rank-biserial effect size.

Workload	n	$\overline{\Delta}$	Median Δ	p	r
HotpotQA	200	+16.38	+0.00	4.6×10^{-7}	+0.67
2WikiMQA	200	+9.89	+0.00	1.8×10^{-4}	+0.64
MuSiQue	200	+13.83	+0.00	3.1×10^{-6}	+0.66

HorizonKV's improvement is statistically significant at $p < 10^{-3}$ on all three workloads, with positive rank-biserial effects ($r = 0.64$–0.67). We compute $r = (W_+ - W_-)/(W_+ + W_-)$ from the sums of ranks of positive and negative nonzero differences, assigning average ranks to ties. Zero

medians reflect frequent equal task scores; zero-difference pairs are excluded from the rank statistic, while mean gaps include all tasks.

F.4 Hardware and Software

All experiments run on a Linux node with 8 NVIDIA A800 80 GB GPUs. The software stack is Python 3.10, PyTorch 2.1, CUDA 12.1.