

SideQuest: Model-Driven KV Cache Management for Long-Horizon Agentic Workloads

Sanjay Kariyappa & G. Edward Suh
NVIDIA
{skariyappa, esuh}@nvidia.com

Abstract

Long-running agentic tasks, such as deep research, require multi-hop reasoning over information distributed across multiple webpages and documents. In such tasks, the LLM context is dominated by tokens from external retrieval, causing memory usage to grow rapidly and limiting decode performance. While several KV cache compression techniques exist for long-context inputs, we find that existing heuristics fail to support multi-step reasoning models effectively. We address this challenge with *SideQuest* – a novel approach that leverages the Large Reasoning Model (LRM) itself to perform KV cache compression by reasoning about the usefulness of tokens in its context. To prevent the tokens associated with this management process from polluting the model’s memory, we frame KV cache compression as an auxiliary task executed in parallel to the main reasoning task. Our evaluations, using a model trained with just 215 samples, show that SideQuest reduces peak token usage by up to 65% and end-to-end latency by 36.8% on agentic tasks with minimal degradation in accuracy, outperforming heuristic-based KV cache compression techniques.

1 Introduction

The evolution of Large Language Models (LLMs) from static conversational interfaces to autonomous agents has catalyzed a shift toward long-running, multi-hop reasoning tasks. In applications such as deep research, automated software engineering, and complex workflow orchestration, models must synthesize information distributed across numerous retrieved documents and intermediate thought traces. This shift has fundamentally changed the memory profile of LLM inference; rather than processing a single prompt, agents maintain a growing context that can span hundreds of thousands of tokens over the course of a single task. The primary bottleneck for scaling these agentic workloads is the Key-Value (KV) cache. The KV cache grows linearly with sequence length, posing two critical challenges for efficient inference. First, it consumes a significant and increasing fraction of GPU memory, which reduces the effective batch size and limits the number of concurrent requests a system can service. Second, as the cache grows, the attention mechanism becomes increasingly memory-bandwidth bound, as the GPU must load a massive volume of KV tensors to generate a single new token (Fu, 2024).

To alleviate these pressures, the community has introduced KV cache compression, which aims to bound the memory footprint by retaining only a subset of the most important tokens in the context. Existing approaches largely rely on fixed heuristics to decide which entries to evict. For instance, techniques like H_2O (Zhang et al., 2023) and Scissorhands (Liu et al., 2023) identify *Heavy Hitters*—tokens that consistently receive high attention scores—and preserve them while pruning others. Other methods, such as SnapKV (Li et al., 2024), automatically identify and retain clusters of key information within local attention windows to preserve essential context. These methods operate on the premise that a token’s past importance is a reliable proxy for its future relevance.

However, we observe that these existing techniques are ill-suited for the dynamic nature of agentic reasoning. Current KV cache compression methods are largely designed for settings where a fixed long-context corpus (e.g., a book or a legal document) is queried by a user.

In contrast, agentic workloads involve a context that evolves over time as the model performs multiple rounds of tool-calling and internal reflection. In these scenarios, heuristic-based pruning is often too blunt a tool; a low-importance token in an early reasoning step may suddenly become critical for a synthesis step ten turns later. The imperfect nature of these heuristics can lead to the premature pruning of important tokens, resulting in reasoning failures that are difficult to debug.

In this paper, we propose *SideQuest*, a novel approach that shifts KV cache management from a fixed heuristic to an intelligent, model-driven process. *SideQuest* equips the model with a mechanism to selectively clear segments of its own KV cache, effectively allowing the model to perform its own memory garbage collection. To ensure that the overhead of this self-management does not interfere with the primary task, we frame KV cache compression as an *auxiliary task* executed in parallel to the main reasoning process (Fig. 1c). This architecture prevents the management tokens from polluting the model’s primary attention window while maintaining a lean, high-fidelity KV cache. We make the following key contributions.

- **Evaluation of Heuristic-Based Methods:** We measure the efficacy of heuristics based KV cache compression techniques on multi-step agentic workloads to show that static importance metrics fail to capture the complex utility of tokens in multi-step reasoning.
- **Model-Driven Memory Management:** We introduce *SideQuest*, a novel framework that empowers Large Reasoning Models (LRMs) to actively manage their own context. By leveraging the model’s semantic understanding of the task state, *SideQuest* performs self-referential KV cache eviction to remove obsolete information.
- **Low-Overhead Parallel Reasoning:** We propose a *shared-context parallel reasoning architecture* that executes auxiliary tasks—such as memory management—concurrently with the main reasoning thread. This design allows for intelligent intervention without polluting the primary context window with management tokens.
- **Training Methodology:** We develop a scalable data synthesis pipeline that uses hindsight analysis on reasoning traces to generate high-quality supervision. This allows us to train the model to recognize stale information without requiring expensive human annotation.

On gpt-oss-20b browser agents evaluated on FRAMES and BrowseComp, *SideQuest* achieves up to **65% reduction** in KV cache usage with negligible degradation in accuracy, establishing a superior efficiency-utility trade-off compared to existing heuristic based techniques.

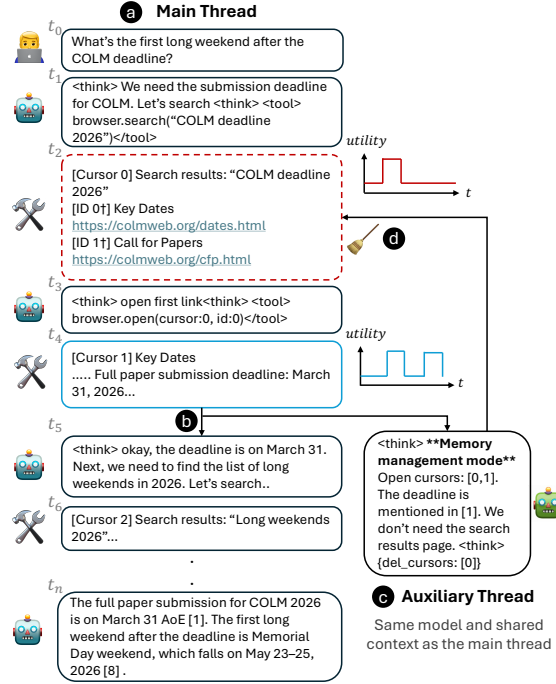


Figure 1: Walkthrough example of *SideQuest* (a) The main thread processes the user request by performing multi-turn reasoning and tool calling. (b) At regular intervals we spawn an auxiliary thread that runs in parallel with the shared context (c) The auxiliary thread reflects on the context and lists the cursors that can be deleted. (d) We clear the messages in the context by invoking a tool, reducing the context size for future turns.

2 Related Work

To mitigate the computational and memory overhead of long-context inference, the community has proposed various architectural and algorithmic optimizations. We provide a brief overview of these techniques in this section. Appendix C has additional related work.

Architectural Optimizations. Architectural innovations aim to reduce the inherent size of the KV state at the model-design level. Grouped-Query Attention (GQA) (Ainslie et al., 2023) and Multi-Head Latent Attention (MLA) (Liu et al., 2024) reduce memory footprint by sharing key and value heads across multiple query heads, significantly lowering the per-token storage cost. Furthermore, alternatives to the standard attention layer, such as Mamba (Gu & Dao, 2024) and Gated Delta Layers (Yang et al., 2024), have been proposed to achieve sub-linear memory growth. These are often interleaved with standard attention layers to create *hybrid models* (e.g. Qwen3-Next (Qwen-Team, 2025), Nemotron-3 (NVIDIA, 2025)) that balance long-range dependency modeling with inference efficiency. While these methods reduce the rate of memory consumption, they do not address the problem of managing the historical context that accumulates during multi-turn agentic reasoning.

Sparse Attention Techniques. Sparse attention aims to reduce the memory bandwidth bottleneck by attending to only a subset of tokens in the context. For instance, DeepSeek Sparse Attention (DSA) (Liu et al., 2025) utilizes a *lightning indexer* to dynamically identify and attend to the most relevant tokens for a given query. While such techniques effectively reduce the pressure on memory bandwidth during the attention computation, they typically do not reduce the physical storage requirements of the KV cache; the full context remains resident in memory, even if only a fraction is accessed during decode.

KV Cache Management and Eviction. Closest to our work are KV cache eviction techniques that prune unimportant entries to bound the memory footprint. StreamingLLM and H_2O (Zhang et al., 2023) utilize static heuristics, such as keeping *attention sinks* or *heavy hitter* tokens based on cumulative attention scores. SnapKV (Li et al., 2024) further optimizes this by identifying key information clusters within an attention window. Recent work has extended these efforts to specifically target reasoning models. R-KV (Cai et al., 2025) leverages the notion of redundancy to determine which reasoning tokens can be safely discarded. RaaS (Hu et al., 2025) introduces the concept of *milestone tokens*—intermediate steps critical for logical progression—and prioritizes their retention. Similarly, LazyEviction (Zhang et al., 2025b) selectively retains thought tokens that demonstrate recurrent importance during the reasoning process, while ThinkKV (Ramachandran et al., 2025) employs a hierarchical approach, categorizing tokens to guide selective quantization and pruning based on their importance levels. Crucially, however, these reasoning-focused methods predominantly target single-step Chain-of-Thought (CoT) tasks, where the context is static and the goal is a single final answer. These methods face a fundamental limitation in agentic settings: *heuristic rigidity*. In multi-step agentic workflows, a token’s importance is highly dynamic. A piece of information that appears irrelevant in turn t (resulting in low attention scores) may become the critical pivot for turn $t + n$. Because existing heuristics lack semantic awareness of the agent’s high-level goals and evolving state, they risk irreversibly pruning information that is vital for downstream reasoning steps.

3 SideQuest

SideQuest is designed for agentic workloads that require multi-step reasoning to resolve complex user queries. These tasks typically employ the ReAct framework, where the model interleaves reasoning traces (thought) with external actions (tool calls) to progressively gather information. To illustrate the memory profile of such tasks, consider the example in Fig. 1. Given a user query about "long weekend after COLM deadline", the model must perform multiple distinct steps: determining the submission date, finding a calendar of holidays, and synthesizing the final answer. Each step generates tool outputs—such as search results or webpage content—that are appended to the context.

Goal. Our primary objective is to identify stale tool calls and associated responses that have lost their utility and evict associated tokens from context as early as possible during the execution of the ReAct agent¹.

Why is this challenging? The utility of a tool response is highly dynamic and non-monotonic, making it difficult to track with simple heuristics. For example, in Fig. 1, the initial search results (Cursor 0) become obsolete the moment the model successfully retrieves the specific webpage containing the conference dates (Cursor 1). It can be safely deleted immediately. In contrast, Cursor 1 undergoes a more complex transition. It is critical to identifying the deadline (Mar 31), but it becomes temporarily irrelevant during the subsequent search for "long weekends". However, unlike the search results, Cursor 1 regains utility at the end of the task when the model cites its sources in the final response. Attention-based heuristics often fail to capture these semantic shifts, leading to either the retention of useless noise (wasteful) or the premature eviction of useful information (harmful).

How SideQuest solves this challenge. Instead of relying on proxy metrics like attention scores, SideQuest leverages the inherent reasoning capabilities of the LRM itself. By analyzing the current state of the ReAct loop and the specific problem definitions, the model explicitly determines which tool responses are no longer needed.

3.1 Overview

SideQuest integrates directly with the ReAct framework to provide a mechanism for self-referential context management. As shown in Figure 1, the architecture operates by periodically spawning an auxiliary thread that executes in parallel with the main thread on the same shared context. The workflow proceeds as follows (see Algorithm 1 for a formal description).

1. **Parallel Execution:** At regular intervals, SideQuest forks the generation process (1b). The auxiliary thread analyzes the history of open tool outputs (e.g., Cursors [0, 1]) relative to the current reasoning step.
2. **Staleness Reasoning:** The model produces a dedicated reasoning trace to determine which inputs are redundant. In the example, it recognizes that because the conference deadline was successfully found in Cursor 1, the search results page (Cursor 0) is no longer required.
3. **Eviction:** The auxiliary thread outputs a structured command, such as `{del_cursors: [0]}`, flagging specific KV cache entries for removal.
4. **Synchronization:** To prevent disrupting the main thread’s generation, the system waits for the current turn of the main thread to complete before evicting the tool responses that are marked for deletion.

Advantages of Parallel Execution. We chose a parallel thread architecture over a sequential approach (performing the user and memory management task in the main thread) for two reasons. First, interleaving memory management steps into the main thread would increase latency, delaying the primary task. Second, explicit management tokens would pollute the context window, partially negating the benefits of compression. By isolating this logic in a transient parallel thread, SideQuest maintains a pristine context for the main agent.

3.2 Steering the Model to Perform Auxiliary Task

In standard multi-turn reasoning, models are conditioned to solve the user’s initial prompt. SideQuest requires a mechanism to override this behavior in the auxiliary thread, shifting the model’s focus from answering the user’s query to managing the context. We achieve this through a two-pronged steering strategy:

1. **Contextual Trigger:** We append a distinct trigger phrase, `** Memory management mode **`, to the start of the auxiliary thread. This serves as a strong indicator to the model that it has entered a maintenance subroutine.

¹We only focus on tool calls and associated responses (referred collectively as “tool response”) as they occupy a significant fraction of the context in the benchmarks that we consider.

2. **Task-Specific Fine-Tuning:** We train the model to recognize this trigger and output the expected deletion commands. This ensures that even when the main context contains complex reasoning about the user’s query, the presence of the trigger guides the model to focus on analyzing the utility of tool responses.

This mechanism allows the **same model** to perform the user’s task in the main thread and memory management in the auxiliary thread, **in parallel** using the **same shared context**.

3.3 Generating Training Data

To equip the model with memory management capabilities without degrading its general reasoning performance, we construct a hybrid training dataset composed of two types of data: *main traces* (for preserving capability) and *auxiliary traces* (for learning eviction). The complete pipeline is formalized in Algorithm 2.

Hindsight Annotation. The core of our data generation relies on hindsight analysis. We start by running inference on a dataset of web-browsing tasks using a base policy (the original model). We filter for traces that lead to correct answers to ensure high-quality supervision. For every tool output (cursor) in a correct trace, we compute its *last-use index* (Line 9). A cursor is considered “expired” at any given turn if it is never referenced again in the future either by tool calls or the final answer².

Main Traces (Distillation). To prevent *catastrophic forgetting* of the model’s primary reasoning skills, we include the original correct traces in the training set. For these samples, we extract the logits from the base policy (Line 10). During training, we apply a logit distillation loss (Hinton et al., 2015) to these entries. This forces the SideQuest model to match the probability distribution of the original model, ensuring that the introduction of memory management tokens does not alter the model’s behavior on standard tasks.

Auxiliary Traces (Cross-Entropy). We synthesize auxiliary traces to teach the model how to identify stale tool responses. At fixed intervals along the trace (Line 15), we identify the set of expired cursors. To simulate realistic inference conditions where the cache is partially compressed, we randomly partition these expired cursors into two sets.

- **Simulated Eviction:** We mask out the tokens corresponding to evicted cursors (Line 16), presenting the model with a context where some eviction has already occurred.
- **Target Generation:** We prompt an annotation model (Line 19) to generate reasoning that explains why the remaining cursors are stale, followed by the requisite deletion command. The prompt used for annotation is provided in Appendix A.

Joint Optimization. The final model is trained on the union of these datasets using a weighted average of the two objectives $\mathcal{L} = \mathcal{L}_{\text{CE}}(\mathcal{D}_{\text{aux}}) + \lambda \mathcal{L}_{\text{distill}}(\mathcal{D}_{\text{main}})$. This joint optimization ensures the model learns to enter the auxiliary mode strictly when triggered, while retaining the robust reasoning capabilities of the base model for the main task.

3.4 Overheads of SideQuest

Storage Costs. The auxiliary thread operates on the shared context of the main thread, meaning it does not duplicate the large KV cache of the conversation history. The only additional storage cost arises from the transient tokens generated during the auxiliary reasoning phase. Once the thread concludes, these tokens are discarded, ensuring that SideQuest introduces zero permanent token overhead to the main context.

Compute and Memory Movement Cost. Executing an auxiliary thread naturally incurs additional computational and memory bandwidth overhead. However, long-context agentic workloads are predominantly bottlenecked by memory bandwidth rather than compute capacity. Although the auxiliary thread imposes a transient cost, we demonstrate empirically that this investment yields a net reduction in resource consumption. By proactively pruning

²This is a simplifying assumption to help with annotation. In reality it is possible that the model uses the information in a tool response without referencing the cursor in some cases.

voluminous tool responses—often spanning hundreds of tokens—SideQuest significantly reduces the cumulative memory movement required for all subsequent reasoning turns.

4 Experimental Setup

Datasets. We focus our evaluation on two long-context, multi-turn web browsing tasks—FRAMES (Krishna et al., 2024) and BrowseComp (Wei et al., 2025)—that require agents to synthesize information across multiple retrieval steps, often resulting in reasoning chains and context windows exceeding 100k tokens (Figure 2). FRAMES evaluates multi-hop reasoning over Wikipedia; to simulate a realistic scale, we evaluate 424 samples against a corpus of 6.4 million Wikipedia articles (Wikimedia). The FRAMES

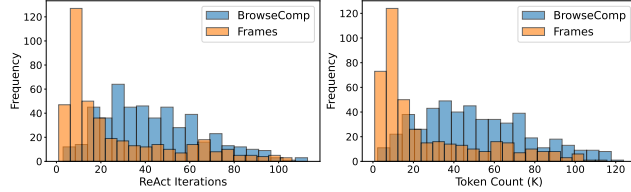


Figure 2: Distribution of ReAct Iterations and token count for FRAMES and BrowseComp with gpt-oss-20b.

rollouts used to construct the training set are drawn from a train partition disjoint from these evaluation examples. BrowseComp tests the ability to navigate the web for specific, hard-to-locate information with short, verifiable answers. To ensure reproducible comparisons, we evaluate 500 samples from BrowseComp using the BrowseComp-Plus (Chen et al., 2025) corpus, which provides a fixed set of 100k documents containing both supporting evidence and challenging negative distractors. Finally, to ensure deterministic evaluation across both benchmarks, we implement a local server backed by these respective corpora that exposes search and retrieval APIs (e.g., `search()`, `open()`) to serve as the backend for the agent’s browser tool.

Models. We conduct experiments using the gpt-oss-20b (OpenAI, 2025) model, which is natively trained with a browser tool that indexes all tool outputs with unique cursor identifiers (e.g., `[Cursor 0]`). This cursor-based indexing allows SideQuest to perform granular, object-level eviction of search results and web/document content. It also facilitates the automated collection of training data for the auxiliary task, as detailed in Section 3.3. Our implementation requires addressable tool-output objects that the runtime can map to KV spans; cursor identifiers provide this interface in the evaluated browser setting. We report results for this model under both *Medium* and *High* reasoning effort configurations.

Training. To construct our training corpus, we sampled 400 queries from the FRAMES (Krishna et al., 2024) dataset. After filtering for traces that resulted in correct answers, we obtained 215 high-quality samples. We then applied the data generation pipeline described in Section 3.3 (Algorithm 2) using gpt-oss-120b as the annotation model $\mathcal{M}$ and an interval of $k = 4$. This process yielded a dataset comprising 215 main traces and 1274 auxiliary traces. To balance the dataset and preserve the model’s core reasoning capabilities, we upsampled the main traces by a factor of $3\times$. We fine-tuned the gpt-oss-20b model using LoRA (Hu et al., 2021) for 3 epochs with a learning rate of 2×10^{-4} and a distillation loss weight of $\lambda = 500$. The LoRA configuration used rank $r = 8$ and $\alpha = 16$. To minimize training overhead while retaining capacity, we applied adapters exclusively to a subset of the projection layers (`gate_up_proj`, `down_proj`) in the Mixture-of-Experts (MoE) layers at depths 7, 15, and 23, keeping all other parameters frozen. Finally, we employed Quantization Aware Training to ensure the final model could be robustly quantized to the `mxfp4` format.

Baselines. We compare SideQuest against an uncompressed baseline (full attention) and three representative KV cache compression techniques. For all heuristic baselines, we evaluate performance at two token budgets: **16k** and **24k** tokens. We additionally evaluate KVzip on a matched FRAMES-medium slice at a 24k prompt budget.

- **H_2O (Zhang et al., 2023).** A “Heavy Hitter” oracle that retains tokens with the highest cumulative attention scores while evicting others. This represents the standard for frequency-based pruning.

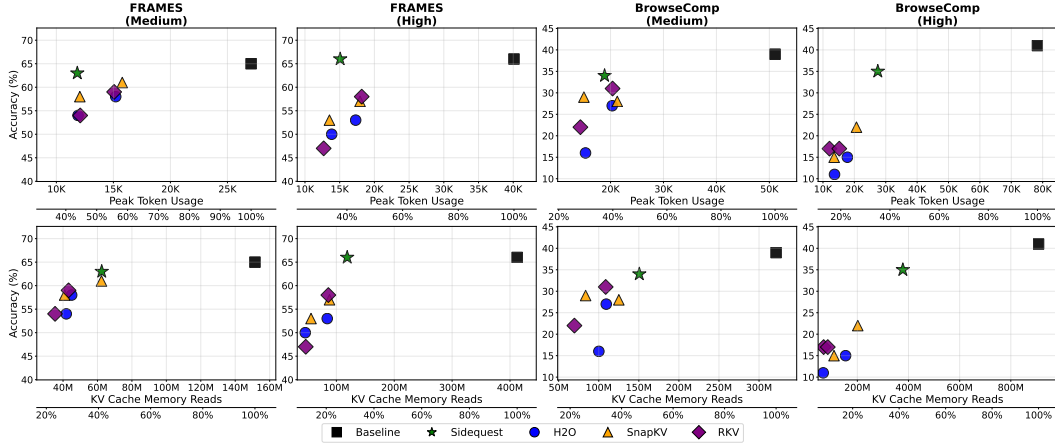


Figure 3: **Efficiency vs. Utility Trade-off.** We evaluate Accuracy against Peak Token Usage and KV cache memory reads for gpt-oss-20b with Medium and High reasoning effort, on the FRAMES and BrowseComp benchmarks. The **Uncompressed Baseline** establishes the upper bound for accuracy but incurs the highest memory cost. **SideQuest** achieves substantial memory savings—reducing peak token usage by 56-65% compared to the baseline—while providing a better accuracy compared to heuristic based methods.

- **SnapKV (Li et al., 2024).** This method identifies and retains clusters of key information within the attention window, aiming to preserve local semantic structure.
- **R-KV (Cai et al., 2025).** A reasoning-focused compression technique that scores tokens based on redundancy, pruning those deemed repetitive or non-essential for the current generation step.
- **KVzip (Kim et al., 2025).** A reconstruction-based, query-agnostic method that selects KV entries independently for each attention head. We evaluate its faithful 24k-prompt configuration on FRAMES-medium.

We implement our compression techniques within the SGLang framework (Zheng et al., 2024), ensuring full compatibility with prefix caching to enable efficient context reuse across multiple ReAct iterations.

Metrics. We evaluate the trade-off between system efficiency and model utility using three primary metrics:

- **Peak Token Utilization:** We report the maximum size of the KV cache reached during the execution of a task. This metric serves as a proxy for the worst-case memory requirement, determining the maximum batch size that can be supported on a given GPU.
- **KV Cache Memory Reads:** We quantify the total volume of data transfer required by the attention mechanism during the decode phase. As agentic workloads are typically memory-bandwidth bound, this metric directly correlates with end-to-end inference latency and system throughput.
- **Accuracy:** We use average task accuracy to assess model utility.

To visualize the cost-benefit trade-off, we plot Utility vs. Efficiency curves across both benchmarks under medium and high reasoning efforts, highlighting the Pareto frontier of memory savings versus reasoning performance.

Serving Metrics. To evaluate the real-world impact on production systems, we additionally report System Throughput (tokens/second), Normalized KV Cache Usage, and Total Benchmark Runtime by implementing SideQuest in SGLang. These metrics capture the practical efficiency gains in a high-concurrency production-grade environment.

5 Results

We present our main experimental results in Figure 3, plotting the trade-off between model utility (Accuracy) and system efficiency (Peak Token Usage and KV Cache Reads).

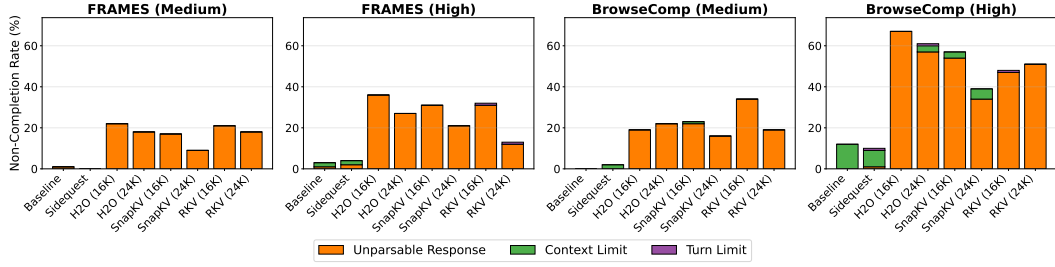


Figure 4: Non-Completion Rate across benchmarks, categorized by failure type: Unparsable Responses (orange), Context Limits (green), and Turn Limits (purple). SideQuest demonstrates superior reliability, matching the near-zero failure rate of the uncompressed baseline, while other methods suffer from high rates of model collapse.

Efficiency vs. Utility. SideQuest shifts the Pareto frontier for agentic memory management. As shown in Figure 3, SideQuest reduces Peak Token Utilization by 56 – 65% and KV cache memory reads by 53 – 71%, compared to the uncompressed baseline. This massive reduction in memory load comes with minimal cost to reasoning performance: we observe only a marginal degradation in accuracy of up to 2% on the in-distribution FRAMES benchmark and a 5% degradation on the out-of-distribution BrowseComp benchmark. In contrast, heuristic baselines like H_2O , SnapKV, and R-KV suffer precipitous drops in accuracy at comparable compression levels, failing to maintain the context fidelity required for complex multi-hop reasoning.

Failure of Fixed Token Budgets. A key finding from our analysis is the inadequacy of fixed-budget compression for agentic tasks. As illustrated in the token distribution histograms in Figure 2, there is significant variance in task difficulty, with token counts ranging from a few thousand to over 120k. Fixed-budget methods (e.g., forcing a 16k window) inevitably fail on the long tail of complex queries, while wasting memory on simple ones. Unlike these rigid approaches, SideQuest selects a trajectory-dependent active context by dynamically evicting cursors it judges no longer semantically relevant.

Robustness and Coherence. Beyond accuracy, we analyze the reliability of the model’s generation process. Figure 4 reports the Non-Completion Rate, which aggregates failures caused by unparsable/non-terminating responses, context length limits, or non-terminating loops. We find that heuristic baselines exhibit a dangerously high rate of unparsable responses (orange bars). This suggests that heuristic pruning often removes tokens critical for syntactic coherence or logic flow, causing the model to produce meaningless responses. SideQuest, by contrast, maintains a non-completion rate comparable to the uncompressed baseline, ensuring that the aggressive memory savings do not compromise the structural integrity of the agent’s reasoning loop. Note that the lower peak memory usage observed for some baselines in Figure 3 (e.g., BrowseComp-High) is partly an artifact of these early crashes; SideQuest achieves its efficiency gains while successfully running tasks to completion.

Serving Efficiency Analysis using SGLang. To validate the real-world impact of our method on production systems, we report various performance metrics using an SGLang-

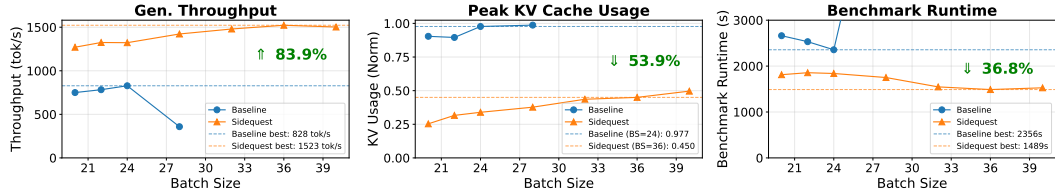


Figure 5: **Serving Performance in SGLang.** Comparison of SideQuest against the uncompressed baseline on the FRAMES benchmark using a single NVIDIA H100 GPU. SideQuest increases peak throughput by 83.9%, reduces KV cache usage by 53.9%, and lowers total benchmark runtime by 36.8%.

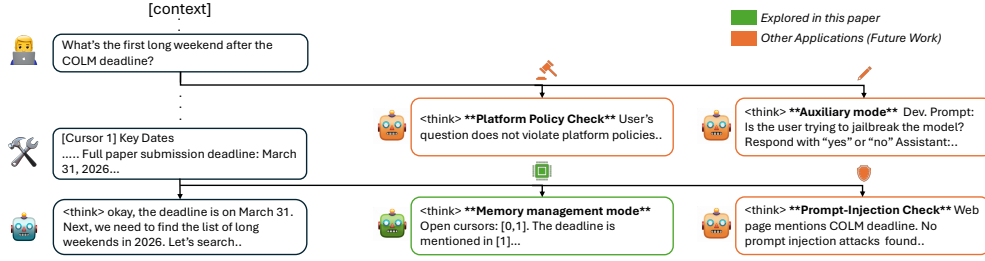


Figure 6: The *SideQuest* framework can be extended to various auxiliary tasks beyond the memory management task.

based implementation of SideQuest. We report system throughput, peak KV Cache usage and total runtime by running 424 samples of the FRAMES benchmark with gpt-oss-20b under medium-effort setting. We report these metrics under different levels of concurrency (batch size). As shown in Figure 5, SideQuest increases peak system throughput by 83.9% (1523 tok/s vs. 828 tok/s) compared to the uncompressed baseline, enabling the engine to scale to larger batch sizes (up to 36) without saturating memory. This performance gain is driven by a massive 53.9% reduction in peak KV cache usage (dropping normalized occupancy from 0.977 to 0.450), which significantly alleviates the memory bandwidth bottlenecks inherent to long-context agentic workloads. Consequently, SideQuest reduces the total end-to-end benchmark runtime by 36.8% (1489s vs. 2356s), demonstrating that our proactive memory management translates directly to faster, more efficient production serving for LRMs. We omit serving metrics for per-head compression baselines due to prohibitive memory relocation overheads, as detailed in Appendix F.

Parallel Auxiliary Thread. To isolate the execution architecture, we compare SideQuest with an inline variant that performs the same memory-management task in the main trajectory. Table 1 reports the mean over three stochastic evaluation runs. The parallel variant achieves higher completion and lower peak active KV usage while keeping memory-management tokens out of the persistent main-thread context. Since the inline variant’s lower accuracy coincides with lower completion, we interpret this result as a reliability and efficiency benefit of the parallel design rather than as an isolated measure of conditional reasoning quality.

Table 1: **Parallel auxiliary-thread ablation** on 424 FRAMES-medium examples. Values are means over three stochastic evaluation runs. Both variants use the same base model, FRAMES traces, auxiliary annotations, and training recipe; they differ only in whether memory management is placed inline or in a parallel thread.

Method	Accuracy (%)	Completion (%)	Peak active tokens
Inline eviction	58.0	90.3	17.60K
SideQuest (parallel)	61.3	99.0	11.51K

6 Limitations

While SideQuest demonstrates significant memory savings, we acknowledge two primary limitations in our current implementation. First, although our method closely matches the uncompressed baseline, we observe minor performance degradation, especially for the out-of-distribution BrowseComp dataset. We hypothesize that this is a data scale issue rather than an architectural flaw. Our current model was fine-tuned on a relatively small dataset constructed from only 215 traces. We believe that scaling the training data to include a larger, more diverse distribution would close this remaining gap. Second, our current eviction strategy is scoped exclusively to tool-responses. Unlike heuristic techniques which can prune any token in the sequence, SideQuest does not yet attempt to compress the agent’s own intermediate reasoning steps. Extending our approach to perform thought pruning

is a promising future direction of research. Additionally, our method operates at a higher level of abstraction compared to prior works on KV compression that rely on attention weights. Exploring a combination of the two is an interesting avenue for future research. Finally, our empirical evidence is limited to gpt-oss-20b with a browser-style cursor schema; generalization to other model families, tool schemas, or domains remains untested.

7 Future Work

New Domains for Memory Management. Our evaluations in this paper are focused on multi-turn web browsing, which serves as an ideal testbed for dynamic context management. Extending SideQuest to other domains remains untested and requires addressable tool outputs that the runtime can map to KV spans. A promising direction for future research is applying this framework to coding agents, where tasks involve traversing massive codebases and reasoning over long-context dependency graphs. The ability to selectively forget irrelevant file contents while retaining critical function definitions could unlock significant efficiency gains.

SideQuest for Auxiliary Tasks. Beyond memory management, SideQuest’s parallel architecture can steer LRMs to execute safety, governance, or custom instruction checks concurrently with the main thread (Figure 6). Instead of relying on separate guardrail models, SideQuest empowers the primary LRM to enforce safety using its full multi-turn context awareness, eliminating the computational overhead of independent context reprocessing.

8 Conclusion

In this work, we demonstrate that static heuristic compression fails to capture the dynamic utility of tokens in long-running browser-agent tasks. We propose SideQuest, a framework for model-driven memory management via a parallel auxiliary thread. This architecture allows for precise, semantic-aware eviction of stale tool outputs without interfering with the primary reasoning process. On gpt-oss-20b browser agents, SideQuest achieves a reduction of up to 65% in peak memory usage with only a minor drop in accuracy, strictly outperforming heuristic baselines. Crucially, our method eliminates the need for manual token budgeting, adaptively scaling context size to match the instantaneous complexity of the query. These results establish model-driven tool-output eviction as an effective approach for the browser-agent setting studied here.

References

- Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.
- Hammad Bashir, Kelly Hong, Patrick Jiang, and Zhiyi Shi. Chroma context-1: Training a self-editing search agent. Technical report, Chroma, March 2026. URL <https://trychroma.com/research/context-1>.
- Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. Cache me if you can: How many kvs do you need for effective long-context lms? *arXiv preprint arXiv:2506.17121*, 2025.
- Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*, 2025.
- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

- Wenjie Du, Li Jiang, Keda Tao, Xue Liu, and Huan Wang. Which heads matter for reasoning? rl-guided kv cache compression. *arXiv preprint arXiv:2510.08525*, 2025.
- Yao Fu. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv preprint arXiv:2405.08944*, 2024.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arxiv* 2021. *arXiv preprint arXiv:2106.09685*, 10, 2021.
- Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. Raas: Reasoning-aware attention sparsity for efficient llm reasoning. *arXiv preprint arXiv:2502.11147*, 2025.
- Jediah Katz. Dynamic context discovery, jan 2026. URL <https://cursor.com/blog/dynamic-context-discovery>. Cursor Blog.
- Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*, 2025.
- Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*, 2026.
- Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananeey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation, 2024. URL <https://arxiv.org/abs/2409.12941>.
- 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.
- Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.
- Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.
- NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for Agentic reasoning, 2025. URL <https://arxiv.org/abs/2512.20848>. Technical report.
- OpenAI. gpt-oss-120b, 2025. URL <https://arxiv.org/abs/2508.10925>.
- Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.
- Qwen-Team. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.

- Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Bruce Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. *arXiv preprint arXiv:2510.01290*, 2025.
- Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.
- Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024.
- Jiayi Tian, Seyedarmin Azizi, Yequan Zhao, Erfan Baghaei Potraghloo, Sean McPherson, Sharath Nittur Sridhar, Zhengyang Wang, Zheng Zhang, Massoud Pedram, and Souvik Kundu. Skipkv: Selective skipping of kv generation and storage for efficient inference with large reasoning models. *arXiv preprint arXiv:2512.07993*, 2025.
- Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.
- Wikimedia. Wikimedia downloads. URL <https://dumps.wikimedia.org>.
- Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.
- Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.
- Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.
- Haoyue Zhang, Hualei Zhang, Xiaosong Ma, Jie Zhang, and Song Guo. Lazyeviction: Lagged kv eviction with attention pattern observation for efficient long reasoning. *arXiv preprint arXiv:2506.15969*, 2025b.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Prompt Used for Generating Auxiliary Traces

We use the following prompt for the annotation model to generate reasoning that is used to construct auxiliary traces as described in Section 3.3.

```
I will provide you with a Task, a Partial Conversation (the history), and the Solution. I want you to create a synthetic reasoning content for the task leads to the provided solution.

Your Objective:
Generate a concise reasoning trace that an agent would think in the moment to distinguish between cursors that are still valuable versus those that are now just noise.

Input Data:
```

```

<Task>
Given a partial conversation, identify the cursors (corresponding to browser tool
calls) that are no longer useful for the assistant and can be removed.
</Task>

<Partial Conversation>
{formatted_conv}
</Partial Conversation>

<Current State>
open_cursors: {open_cursor_ids}
</Current State>

<Target Solution>
removable_cursors: {del_cursor_ids}
</Target Solution>

<FinalResponseCitations>
cursors_final: {cursors_final}
(These are the cursors cited in the agent's final answer. They are by definition "
useful" and must not be removed.)
</FinalResponseCitations>

**Guidelines for Reasoning:**
1. Do not mention that you have the <Target Solution> OR <FinalResponseCitations> in
the reasoning. Reason based *only* on the conversation history up to this
point. It's okay to say that a cursor is needed for citation in the final
answer.
2. The reasoning should first reflect on all the open cursors and only then
determine which cursors are no longer useful.
3. Reasoning should be **concise**.
4. If `<Target Solution>` is an empty list, all currently open cursors are useful
and no cursors can be removed.

**Response Format:**
Respond with the following JSON structure:
{
  "reasoning": "...",
  "useful_cursors": [<id>, <id>],
  "removable_cursors": [<id>, <id>]
}

```

B Examples of Memory Management Reasoning

Due to the long nature of the agentic tasks we cannot fit an entire trace within a page. However, we provide some examples of the auxiliary trace produced by SideQuest in Fig. 7 to demonstrate the sophisticated reasoning used in KV cache management.

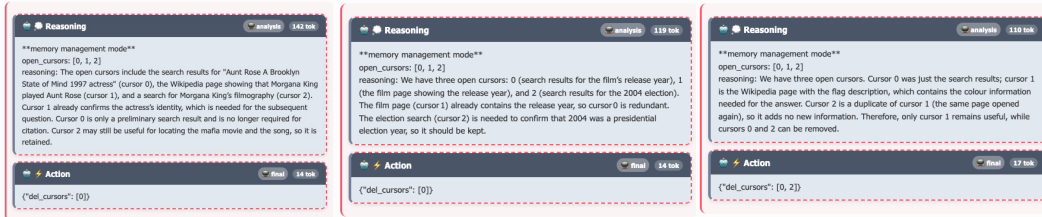


Figure 7: Examples of memory-management reasoning produced by SideQuest.

C Additional Related Work

C.1 KV Cache Compression and Reasoning-Trace Eviction

Adjacent work studies KV compression at sentence, token, or attention-head granularity. SkipKV (Tian et al., 2025) removes redundant sentence spans in single-stream Chain-of-Thought reasoning. KVzip (Kim et al., 2025) and Fast KVzip (Kim et al., 2026) score individual KV entries using reconstruction- and gating-based objectives for general long-context inference. Head-level methods such as DuoAttention, RazorAttention, RLKV, and PruLong (Xiao et al., 2024; Tang et al., 2024; Du et al., 2025; Bhaskar et al., 2025) retain full KV state for selected heads while compressing others. These methods are complementary to SideQuest: rather than selecting sentence spans, token/KV pairs, or heads, SideQuest makes task-state-aware decisions over complete, addressable tool-response objects in multi-turn agent trajectories.

C.2 Operating System and Retrieval-Based Approaches

Recent works have proposed moving beyond static context windows by drawing inspiration from operating systems. The most prominent example, MemGPT (Packer et al., 2023), creates a hierarchical memory architecture where the model explicitly manages its own context by swapping text between its active prompt (Main Context) and external storage (External Context) via function calls. Similarly, in the domain of automated software engineering, Cursor has introduced *Dynamic Context Discovery* (Katz, 2026), which applies virtualization principles to code generation. Rather than stuffing the context window with static tool definitions and history, Cursor abstracts these elements as files that the agent can lazily load only when necessary.

While both MemGPT and Cursor solve the information retrieval problem (deciding what text to show the model), they do not address the inference efficiency problem of linear context growth. Even with dynamic discovery, once information is loaded, it occupies GPU memory linearly. SideQuest complements these methods by operating on the internal state: it allows the model to “garbage collect” the heavy tensors of intermediate reasoning steps without breaking the continuity of the task.

C.3 Hierarchical and Decomposition-Based Context Management

Another class of approaches addresses the long-context challenge by restructuring agentic reasoning into hierarchical or recursive formats. Recursive Language Models (RLMs) (Zhang et al., 2025a) treat the context as an external environment, allowing the model to write code that inspects, slices, and recursively calls itself on specific data chunks. Similarly, Context Fold (Sun et al., 2025) introduces a dynamic *branch-and-fold* mechanism where agents can spawn temporary sub-trajectories for specific sub-tasks; upon completion, the entire sub-trajectory is *folded* into a concise summary, and the intermediate tokens are discarded.

Complementarity with SideQuest. These approaches mitigate context growth by isolating intermediate reasoning steps into separate, transient contexts (child processes or branches) and propagating only the final result to the main thread. However, they do not solve the fundamental problem of linear context growth within a specific sub-task. A complex sub-problem (e.g., “fix a failing test”) will still accumulate a massive local context of tool outputs and retrieval artifacts before a result can be returned. SideQuest addresses this orthogonal challenge. By operating as a garbage collector for the active linear stream, SideQuest can be deployed inside the sub-tasks of an RLM or Context Fold architecture. It ensures that the local context of each branch remains lean by surgically evicting stale tool outputs (observations) while preserving the necessary tool outputs. Consequently, SideQuest is highly compatible with structural decomposition methods, offering a mechanism to maximize the efficiency of the individual workers within a hierarchical system.

C.4 Model Driven Memory Management

After the public release of our manuscript, Chroma released Context-1 (Bashir et al., 2026), a self-editing search agent that modifies textual search context. We view it as concurrent work: SideQuest instead uses a parallel auxiliary thread to identify stale tool-response objects and evict the corresponding KV spans at serving time.

D Additional Experiments

D.1 Matched KVzip Comparison

On the same 424-example FRAMES-medium slice, the uncompressed baseline attains 65.0% accuracy (274/424), 99.0% completion, and 27.07K peak active tokens. KVzip with a 24k prompt budget attains 47.0% accuracy (201/424), 58.0% completion, and 16.06K peak active tokens. SideQuest attains 62.0% accuracy (263/424), 99.0% completion, and 11.70K peak active tokens on this slice. KVzip’s 42% non-completion rate arises from malformed Harmony/tool-call traces after KV eviction. We report this matched comparison separately rather than overlaying it on Figure 3, whose points come from the original paper evaluation snapshot.

D.2 Memory Savings by Trajectory Correctness

To test whether the measured memory savings are driven only by unusually long baseline failures, we perform a post-hoc analysis of the same 424 FRAMES-medium trajectories, split by answer correctness. Nearly all trajectories complete, so incorrect trajectories are the most direct indicator of unproductive long generations. Table 2 reports active-KV and decode-cost metrics for all, correct, and incorrect trajectories; Table 3 reports the full bucketed peak-token distribution.

Table 2: **Memory costs split by trajectory correctness** on 424 FRAMES-medium trajectories. Peak and average tokens refer to active KV tokens.

Method	Split	n (%)	Peak	Avg.	KV reads	>32K
Baseline	All	424 (100)	30.70K	14.86K	185.44M	154
Baseline	Correct	260 (61)	17.87K	8.99K	55.81M	40
Baseline	Incorrect	164 (39)	51.04K	24.15K	390.94M	114
SideQuest	All	424 (100)	11.70K	6.05K	64.42M	4
SideQuest	Correct	263 (62)	8.98K	4.85K	23.24M	0
SideQuest	Incorrect	161 (38)	16.14K	8.01K	131.69M	4

Table 3: **Bucketed peak active-KV tokens** by trajectory correctness. Entries are trajectory counts.

Method	Split	<8K	8–16K	16–24K	24–32K	>32K
Baseline	Correct	86	90	30	14	40
Baseline	Incorrect	8	23	10	9	114
SideQuest	Correct	131	118	12	2	0
SideQuest	Incorrect	18	71	53	15	4

The split confirms that incorrect baseline trajectories are substantially longer: 114/164 (70%) exceed 32K peak active tokens, compared with 40/260 (15%) correct baseline trajectories. However, the improvement is not limited to these outliers. On correct trajectories, SideQuest reduces mean peak active tokens from 17.87K to 8.98K and decode KV-read cost from 55.81M to 23.24M. It also reduces these metrics on incorrect trajectories (51.04K to 16.14K peak tokens and 390.94M to 131.69M KV reads).

D.3 Comparison with Oldest-First Eviction Policy

SideQuest enables the model to dynamically reason about its context, making semantic decisions about *when*, *which*, and *how many* tool responses to evict. In many agentic workflows the older tool responses lose utility first. This raises a natural baseline question: is a simple First-In-First-Out (FIFO) heuristic with a fixed token budget sufficient?

To evaluate this, we compare SideQuest against an "oldest-first" static policy. We impose fixed context budgets of 7K, 9K, and 11K tokens; whenever the context exceeds this limit, the oldest tool response is evicted. We measure performance across 64 samples from the FRAMES dataset. The results, detailed in Figure 8, demonstrate several critical failures of the static heuristic:

- **Degraded Utility at Iso-Memory:** At roughly iso-active token counts (SideQuest averages 6002 tokens, while the 7K and 9K static budgets average 5599 and 6343 tokens, respectively), the static policy’s accuracy drops sharply to 52.0% and 53.0%. This significantly underperforms both the uncompressed baseline (58.0%) and SideQuest (66.0%).
- **Action Churn and Inefficiency:** The static policies consistently trigger a higher number of ReAct iterations (e.g., 47.0 average turns for the 7K budget compared to 31.7 for SideQuest). This inflation occurs because the agent is frequently forced to re-query webpages that were prematurely evicted by the blind FIFO policy. Consequently, despite processing fewer tokens per forward pass, the repeated generation steps increase the total benchmark runtime (e.g., 444s for the 7K budget vs. 375s for SideQuest).
- **Model Instability:** Finally, the rigid eviction policy severely degrades generation reliability. While SideQuest maintains a 0.0% non-completion rate (perfectly matching the baseline), the oldest-first policies exhibit high failure rates, peaking at 16.0% for the 7K budget. This indicates that arbitrarily deleting early context often removes critical structural formatting or original task constraints, leading to unparseable outputs or non-terminating loops.

Ultimately, these results confirm that token utility in multi-step reasoning is non-monotonic; dynamic, semantic eviction is necessary to compress context without breaking the agent’s logic.

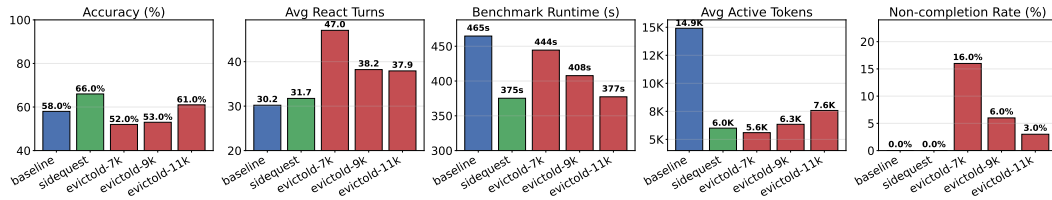


Figure 8: Comparing SideQuest with a static algorithm (*evictold*) that evicts the oldest tool response when the context window exceeds the token budget.

D.4 Training and Sensitivity Ablations

We evaluate training choices on the same 424-example FRAMES-medium slice. Tables 4–7 report accuracy, active-KV, and decode-cost metrics. The paper setting uses $k = 4$, $\lambda = 500$, gpt-oss-120b annotations, and $3 \times$ main-trace upsampling.

The $\lambda = 100$ –1000 settings preserve similar utility and large memory reductions, while $\lambda = 0$ collapses and $\lambda = 5000$ suppresses the learned eviction behavior. Triggering at $k = 1$ –4 yields comparable savings, whereas $k = 6$ retains more stale state and has lower accuracy. The annotation-size and non-collapsed upsampling ablations are similarly stable. For LoRA placement, we use the gpt-oss fine-tuning cookbook configuration, targeting expert projections (gate_up_proj and down_proj) in MoE layers 7, 15, and 23; this choice was not tuned for SideQuest.

Table 4: **Distillation-weight sweep.** The $\lambda = 0$ model collapses, so its token and cost values are not meaningful efficiency measurements.

Setting	Accuracy (%)	Peak	Avg.	KV reads
Baseline	65.0	27.07K	13.06K	151.45M
$\lambda = 0$	0.0	0.52K*	0.48K*	0.02M*
$\lambda = 100$	59.0	11.96K	6.22K	71.70M
$\lambda = 500$ (paper)	62.0	11.70K	6.05K	64.42M
$\lambda = 1000$	60.0	12.60K	6.27K	72.18M
$\lambda = 5000$	57.0	29.78K	14.31K	209.08M

Table 5: **Auxiliary-trigger interval sweep.** All SideQuest settings reduce active KV relative to the uncompressed baseline.

Setting	Accuracy (%)	Peak	Avg.	KV reads
Baseline	65.0	27.07K	13.06K	151.45M
$k = 1$	62.0	10.69K	5.35K	69.17M
$k = 2$	63.0	11.49K	5.66K	69.25M
$k = 3$	62.0	11.40K	5.74K	61.21M
$k = 4$ (paper)	62.0	11.70K	6.05K	64.42M
$k = 6$	57.0	13.23K	6.70K	67.88M

E Algorithms

This section provides the formal pseudocode for the core components of the SideQuest framework.

Algorithm 1 (Operation of SideQuest) details the runtime inference procedure. It outlines the parallel execution of the main ReAct reasoning loop and the periodic spawning of the auxiliary thread, demonstrating how context management and eviction are synchronized without interrupting the primary task.

Algorithm 2 (Training Data Pipeline) describes our offline data synthesis methodology. It formalizes the hindsight annotation process used to identify expired tool responses and details how both main and auxiliary training traces are constructed to teach the model memory management while preserving its core reasoning capabilities.

F Serving Overhead of Per-Head KV Cache Compression

In Section 5, we report serving throughput and runtime only for SideQuest and the uncompressed baseline. We omit these metrics for H2O, SnapKV, and R-KV because they retain different token subsets per KV head, whereas SGLang’s attention backend uses one shared `kv_indices` array for all heads. Our implementation must therefore relocate each head’s selected KV entries into common destination slots. For gpt-oss-20b (24 layers and 8 KV heads), each compression event performs $24 \times 8 = 192$ per-(layer, KV-head) indexed relocations, each moving both key and value entries through temporary buffers. This data movement results in lower end-to-end throughput than the uncompressed baseline. SideQuest instead evicts complete tool-response spans identically across heads and layers, requiring no per-head relocation.

Table 6: **Annotation-model-size ablation.** Both models produce comparable utility and memory savings.

Annotator	Accuracy (%)	Completion (%)	Peak	Avg.	KV reads
gpt-oss-20b	62.0	99.0	11.50K	5.79K	59.76M
gpt-oss-120b	62.0	99.0	11.70K	6.05K	64.42M

Table 7: **Main-trace upsampling sweep.** The $0\times$ model collapses, so its token and cost values are not meaningful efficiency measurements.

Upsampling	Accuracy (%)	Completion (%)	Peak	Avg.	KV reads
$0\times$	0.0	85.0	0.52K*	0.48K*	0.02M*
$1\times$	62.0	96.0	11.51K	6.10K	66.20M
$3\times$ (paper)	62.0	99.0	11.70K	6.05K	64.42M
$6\times$	62.0	100.0	12.14K	6.18K	68.57M

G LLM Usage

In accordance with conference guidelines, we disclose the use of Large Language Models (LLMs) in the preparation of this manuscript. The core research ideas, experimental design, data generation and evaluation were conducted entirely by the authors. We utilized an LLM as an editorial assistant to help refine prose, condense text for length constraints, and assist in drafting and formatting specific subsections based strictly on the empirical results and detailed outlines provided by the authors.

Algorithm 1 Operation of SideQuest

Require: User Query Q , Model $\mathcal{M}$, Trigger Interval K , Trigger Phrase p

```

1: Initialize context  $\mathcal{C} \leftarrow [Q]$ 
2: Initialize turn counter  $t \leftarrow 0$ 
3: Initialize auxiliary thread  $S_{\text{thread}} \leftarrow \text{None}$ 
4: while True do
5:   // Phase 1: Eviction
6:   if  $S_{\text{thread}} \neq \text{None}$  and  $S_{\text{thread}}.\text{is\_finished}()$  then
7:      $\Delta_{\text{ids}} \leftarrow S_{\text{thread}}.\text{get\_output}()$ 
8:      $\mathcal{C}.\text{clear\_kv}(\Delta_{\text{ids}})$  // Prune stale tool outputs from cache
9:      $S_{\text{thread}} \leftarrow \text{None}$ 
10:  end if
11:  // Phase 2: Auxiliary Thread Spawning
12:  if  $t \pmod K = 0$  and  $S_{\text{thread}} = \text{None}$  then
13:     $\mathcal{C}_{\text{aux}} \leftarrow \mathcal{C} + p$  // Append trigger phrase
14:     $S_{\text{thread}} \leftarrow \text{ASYNCGENERATE}(\mathcal{M}, \mathcal{C}_{\text{aux}})$ 
15:  end if
16:  // Phase 3: Main ReAct Execution
17:   $R \leftarrow \mathcal{M}(\mathcal{C})$ 
18:   $\mathcal{C} \leftarrow \mathcal{C} + R$ 
19:  if  $R$  contains FINAL ANSWER then
20:    return  $R$ 
21:  end if
22:  if  $R$  contains Tool Call  $T$  then
23:     $O \leftarrow \text{EXECUTETOOL}(T)$ 
24:     $\mathcal{C} \leftarrow \mathcal{C} + O$ 
25:  else
26:    break //End of turn if no tool called
27:  end if
28:   $t \leftarrow t + 1$ 
29: end while

```

Algorithm 2 Training Data Pipeline

Require: Dataset $\mathcal{D}$, base policy π , annotation model $\mathcal{M}$, interval k , trigger phrase p

```

1:  $\mathcal{D}_{\text{train}} \leftarrow \emptyset$ 
2: for each task  $x \in \mathcal{D}$  do
3:    $\tau \leftarrow \text{RUNINFERENCE}(\pi, x)$ 
4:   if  $\text{ISCORRECT}(\tau) = \text{false}$  then
5:     continue
6:   end if
7:   // Annotate each cursor with its last-use turn index
8:   for each cursor  $c$  in  $\tau$  do
9:      $\ell_c \leftarrow \text{LASTUSEINDEX}(\tau, c)$ 
10:  end for
11:  // Add main trace with full attention and logits
12:   $\mathbf{z} \leftarrow \pi(\tau)$  // Extract logits from base policy
13:   $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(\tau, \mathbf{I}, \mathbf{z}, \text{main})\}$ 
14:  // Generate auxiliary traces at intervals
15:  for each reasoning turn  $t$  where  $t \bmod k = 0$  do
16:     $\mathcal{C}_{\text{expired}} \leftarrow \{c : \ell_c < t\}$  // Cursors past last-use
17:     $\mathcal{C}_{\text{open}}, \mathcal{C}_{\text{closed}} \leftarrow \text{RANDOMPARTITION}(\mathcal{C}_{\text{expired}})$ 
18:     $\mathbf{M} \leftarrow \text{BUILDMASK}(\tau_{1:t}, \mathcal{C}_{\text{closed}})$  // Mask out closed cursors
19:     $r \leftarrow \mathcal{M}(\tau_{1:t}, \mathcal{C}_{\text{open}}, \mathcal{C}_{\text{closed}})$  // Generate reasoning
20:     $r \leftarrow p \oplus r$  // Prepend trigger phrase
21:     $\tau_{\text{aux}} \leftarrow \tau_{1:t} \oplus [r, \text{CLOSEACTION}(\mathcal{C}_{\text{open}})]$ 
22:     $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(\tau_{\text{aux}}, \mathbf{M}, \emptyset, \text{aux})\}$ 
23:  end for
24: end for
25: return  $\mathcal{D}_{\text{train}}$ 

```
