SEARCHTRIMMER: SEARCH-SELECTIVE KV EVICTION AND LOOP GATING FOR EFFICIENT DEEP RESEARCH AGENTS

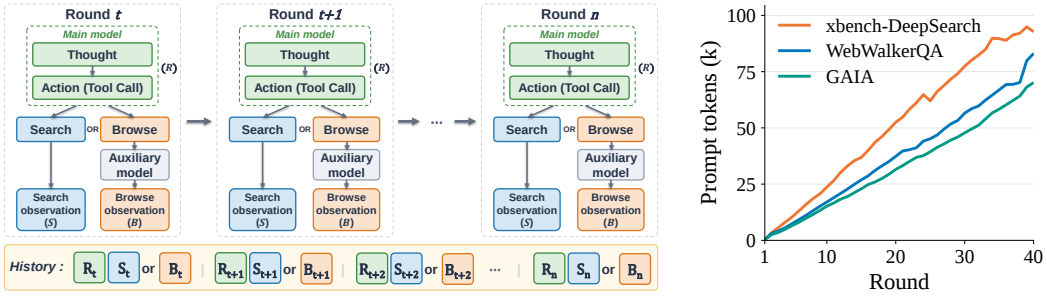
Anonymous authors
Paper under double-blind review

ABSTRACT

Deep research agents incur costs from both an expanding history and repeated rounds of model inference and tool use. The growing history increases KV-cache memory, while compressing it can add rounds as agents attempt to recover missing observations. We propose SearchTrimmer, a training-free framework that addresses both costs through two components motivated by an analysis of agent trajectories. First, search observations occupy most of the history while containing little unique evidence and remaining inexpensive to recover. We therefore propose *Search-Selective KV Eviction*, which retains browse observations and reasoning while keeping only the k most recent search observations in the KV cache. Second, additional search activity often repeats earlier queries. We distinguish two types of re-search: *Restore*, which leads to a newly retained browse observation, and *Loop*, which does not. Since Loops account for most of the increase, we propose *Loop Gating*, which limits their cumulative count and requests a final answer when the budget is reached, while leaving Restores unpenalized. Neither component requires retraining, an additional model, or access to attention scores. Across three benchmarks, SEARCHTRIMMER reduces average KV-cache usage by 41–61% on WebExplorer-8B and 32–36% on WebSailor-32B, with maximum KV-cache reductions of 73–83% and 29–31%, respectively, while matching or exceeding the uncompressed agents' avg@3 accuracy and keeping round counts within approximately 6% of their baselines.

1 INTRODUCTION

Large language models (LLMs) (Zhang et al., 2022; Touvron et al., 2023; Grattafiori et al., 2024; Yang et al., 2025) increasingly augment their parametric knowledge with external tools. The ReAct formulation (Yao et al., 2023) made this practical by interleaving reasoning with tool calls, allowing the model to choose each action based on what the tools have returned. Recent deep research



(a) Round structure and accumulated history

(b) Prompt growth by round

Figure 1: Deep research agent trajectory and prompt growth. **(a)** Each round produces a thought, a tool call, and a search or browse observation, which remain in the history. **(b)** Prompt tokens by round on WebExplorer-8B, averaged over uncompressed trajectories that reach each round.

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.

agents (Team et al., 2025; Liu et al., 2025; Wu et al., 2026a; Tao et al., 2026) adopt this ReAct-style design and use two primary tools, search and browse, over multiple rounds to gather evidence for an answer (Figure 1a). Every thought, tool call, and observation remains in the prompt of subsequent rounds, so the history grows throughout the trajectory (Figure 1b). Since the KV cache stores this growing prompt, its memory usage increases with trajectory length.

This growing history can be reduced using existing methods such as static compression of long inputs (Pan et al., 2024; Liskavets et al., 2025; Li et al., 2024), history summarization (Wu et al., 2025b; Kang et al., 2025), observation windows (Yang et al., 2024; Lindenbauer et al., 2025), and agent-aware KV-cache compression (Matam & Kim, 2026; Huang et al., 2026). However, these approaches either overlook cross-round dependencies, require retraining or an additional model, or apply recency- or model-based retention without distinguishing the roles of different observations. More importantly, compression can change the agent's behavior after information is removed. The agent may re-issue tool calls to recover missing observations, adding rounds of tool execution, prefill, and decode. Some of these calls recover useful evidence, while others merely repeat earlier searches. Efficient compression must therefore reduce both history growth and the additional rounds induced by compression, yet existing methods primarily address the former.

In this paper, we propose SearchTrimmer, a training-free framework that jointly reduces KV-cache usage and compression-induced round overhead in deep research agents. Our analysis reveals two key properties of agent trajectories. First, their histories separate into three content types: search observations, browse observations, and reasoning, which differ substantially in size, evidence density, and recovery cost. Second, repeated searches after eviction differ in whether they recover new retained evidence. These findings motivate two components:

- **Search-Selective KV Eviction**, which retains browse observations and reasoning while evicting older search observations from the KV cache. Search observations dominate the history but contain little unique evidence and are inexpensive to recover, while KV-level eviction preserves information already incorporated into later cache entries.
- **Loop Gating**, which distinguishes re-search that leads to a newly retained browse observation (*Restore*) from re-search that does not (*Loop*). Since Loops account for most of the increase in re-search after eviction, the gate counts only Loops and requests a final answer once their cumulative count reaches a budget.

Neither component requires agent retraining, an additional model, or access to attention scores or hidden states at serving time. Together, they reduce KV-cache usage and repeated tool calls while preserving the accuracy of the uncompressed agent.

2 Background

2.1 Long-Horizon Deep Research Agents

A deep research agent answers a question by interacting with web tools over multiple rounds (Team & Inc, 2026). Deep research agents typically use two core tools, search and browse. Search returns ranked results for a query, each with a title, URL, and snippet, while browse takes a URL and a goal and returns a distilled answer from the fetched page through an auxiliary model.

Recent agents commonly follow a ReAct-style design (Wu et al., 2026a; Li et al., 2025; Tao et al., 2026; Liu et al., 2025), where each round consists of a thought, a tool call, and the resulting observation, except the final round, which produces the answer. The sequence of rounds forms a *trajectory*, while their accumulated content forms the *history*. The history separates into three content types: **search observations**, **browse observations**, and **reasoning**. Search and browse observations are returned by the corresponding tools, while reasoning consists of the thought and tool call produced by the main model.

A trajectory incurs two main costs. First, the history is carried forward in the prompt of each subsequent round, so KV-cache usage grows as the history accumulates. On WebExplorer-8B, prompt length increases by 1.7k–2.5k tokens per round depending on the benchmark. Second, each round adds a tool call, prefill, and decode. Efficient serving therefore requires controlling both KV-cache growth and the number of rounds.

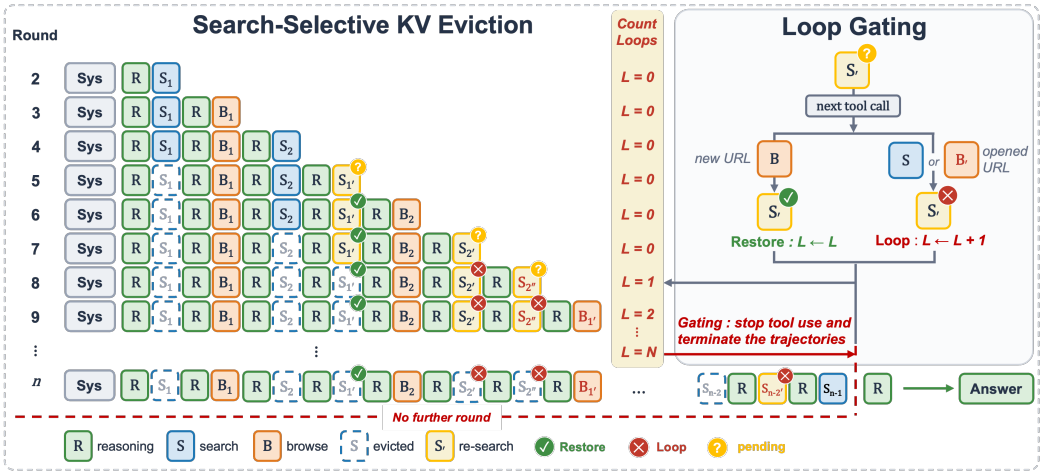


Figure 2: The method on one trajectory. Left, *Search-Selective KV Eviction* with $k = 2$: dashed boxes denote evicted search observations, while the two most recent are retained in the KV cache. Right, *Loop Gating*: each re-issued query is resolved by the next tool call as a *Restore* if it opens a new page, and as a *Loop* otherwise. The trajectory terminates when the cumulative Loop count reaches N, before the next proposed tool call is executed.

2.2 Context Compression for Long-Horizon Agents

Static compression of long inputs. Static compressors reduce a long input either before encoding (Pan et al., 2024; Liskavets et al., 2025) or within the KV cache (Li et al., 2024; Liu et al., 2026; Jo et al., 2026). CPC (Liskavets et al., 2025) removes sentences with low question relevance, while ChunkKV (Liu et al., 2026) retains chunks with high attention from a recent window. FastKV (Jo et al., 2026) similarly selects tokens using recent-window attention at an intermediate layer and propagates only the selected tokens to later layers, reducing both prefill compute and KV memory. These methods are designed for a single input and generation, whereas a deep research agent performs multiple generations across rounds. Applying them independently at each round may therefore ignore cross-round dependencies and discard information important to the overall trajectory.

History summarization. History summarization methods rewrite the accumulated history into a compact representation. Some train the main model to compress its own history through reinforcement learning (Zhou et al., 2026; Wu et al., 2026b) or supervision (Ye et al., 2025; Lu et al., 2026; Li et al., 2026), requiring retraining. Others keep the main model unchanged but add a separate compressor (Wu et al., 2025b; Kang et al., 2025). These approaches therefore incur either training cost or additional serving overhead.

Observation window. A training-free alternative removes older observations rather than rewriting them. SWE-agent (Yang et al., 2024) replaces observations outside a recent window with one-line placeholders while retaining thoughts and actions. Lindenbauer et al. (2025) show that this simple rule achieves comparable solve rate and cost to LLM-based summarization without requiring an additional model. However, retention is determined by recency across all observations, treating observations with different roles as interchangeable. Moreover, these methods do not distinguish or control the additional rounds induced by compression.

Agent-aware KV cache compression. Recent methods directly compress the KV cache as the agent trajectory grows (Matam & Kim, 2026; Huang et al., 2026). However, they compress to a predefined KV budget, which does not account for the wide variation in trajectory length across questions. Moreover, neither method explicitly controls the additional rounds induced by eviction.

3 Method: SearchTrimmer

Our method has two parts (Figure 2). Part 1 uses *Search-Selective KV Eviction*, motivated by differences in size, evidence density, and recovery cost across history content types, to retain browse

Table 1: Evidence held by each content type at the final round, over the correct rollouts (Appendix A). Share is the fraction of tokens across the three types, sole is the fraction of facts held by that type only, and density is sole divided by share.

	WebExplorer-8B			WebSailor-32B			
Type	Share	Sole	Density	Share	Sole	Density	Recovery cost
Search	88.4%	4.3%	0.05	71.3%	1.7%	0.02	one round
Browse	3.3%	6.4%	1.93	11.2%	14.4%	1.29	one round + auxiliary model call
Reasoning	8.4%	1.1%	0.13	17.4%	0.1%	0.01	not reproducible

observations and reasoning while evicting older search observations. Part 2 uses *Loop Gating* to control the additional rounds induced by eviction by distinguishing re-search that recovers new retained evidence from re-search that does not.

3.1 Part 1: Search-Selective KV Eviction

3.1.1 Motivation: What the History Holds

We analyze trajectories from two deep research agents, WebExplorer-8B (Liu et al., 2025) and WebSailor-32B (Li et al., 2025). We compare their three history content types, search observations, browse observations, and reasoning, in size, evidence density, and recovery cost. Detailed statistics are provided in Appendix A.

Search observations as the eviction target. As shown in Figure 3, search observations dominate the final-round prompt in both agents, making them the primary source of potential memory savings. However, they contain relatively little unique evidence. Table 1 shows that search observations account for 88.4% and 71.3% of the tokens but uniquely hold only 4.3% and 1.7% of the facts, whereas browse observations account for 3.3% and 11.2% of the tokens while uniquely holding 6.4% and 14.4%. Accordingly, evidence density is much lower for search than for browse, with the same pattern across benchmarks (Table 6). Search observations are also inexpensive to recover. Re-acquiring one requires only a re-issued query, with a median latency of 3.5 s in our WebExplorer-8B traces, whereas re-acquiring a browse observation requires a URL fetch and another auxiliary-model pass, with a median latency of 24 s. Reasoning is not reproducible, as it is generated from the history available at that round. Search observations therefore offer the largest memory saving while carrying little unique evidence and remaining inexpensive to recover.

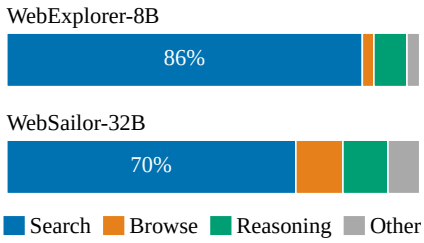
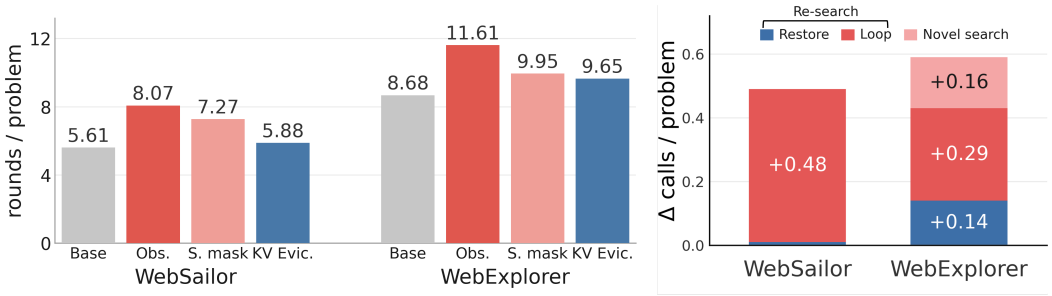


Figure 3: Final-round prompt composition by content type. Other includes the prefix and chat formatting.

Which search observations to keep. Having identified search observations as the eviction target, we next ask which ones should be retained. A search observation can remain useful in two ways: as *evidence*, by containing information relevant to the final answer, and as a *pointer*, by providing URLs that the agent may later browse. As evidence, search observations are typically absorbed quickly into later reasoning or browse observations. More than 75% of facts appearing in search also appear in the next thought or browse observation, and more than 90% within two rounds (Table 7). Keeping only the most recent search observation while retaining browse observations and reasoning leaves no fact missing from the final-round prompt (Table 8). As pointers, recent search observations must remain available while the agent selects pages to browse. One search followed by three or more browse calls occurs in 8.6% and 17.7% of trajectories (Table 9), so a window over all observations may evict the latest search observation before it is fully used. Among browsed URLs originating from search results, the most recent search observation contains the URL in 95.4% and 93.9% of cases, while the two most recent cover 98.8% and 98.5% (Table 10). URLs found only in older search observations can still be recovered by re-issuing the corresponding query at the cost of an additional round.



(a) Rounds per problem (b) Additional search calls

Figure 4: Effect of history compression on rounds and search calls. **(a)** Rounds per problem for the uncompressed baseline (Base), SWE-agent-style observation masking (Obs.), search masking (S. mask), and Search-Selective KV Eviction (KV Evic.). Both masking methods modify the prompt, whereas our method evicts search observations from the KV cache. **(b)** Additional search calls under Search-Selective KV Eviction relative to Base, decomposed into Restore, Loop, and novel search.

Together, these results show that search observations lose utility as they age, while the most recent few capture nearly all evidence and pointers needed by the agent.

3.1.2 Proposed Method: Search-Selective KV Eviction

Based on the analysis above, we retain only the k most recent search observations while preserving all browse observations and reasoning. Recency is measured among search observations rather than across all observations. Once a search observation falls outside the most recent k, it is evicted from the KV cache and is not restored. Formally, let p denote the fixed prefix containing the system prompt and question, and let the trajectory up to round t be $H_t = (p, r_1, o_1, \ldots, r_t, o_t)$, where r_s is the reasoning and o_s is the observation at round s. Let $\mathcal{S}_t$ denote the rounds containing search observations, and let $\mathcal{S}_t^k \subseteq \mathcal{S}_t$ denote the k most recent among them. After round t, we retain the prefix, all reasoning, all browse observations, and the k most recent search observations:

$$\mathcal{K}_t = \{p\} \cup \{r_s : s \le t\} \cup \{o_s : s \le t,\ s \notin \mathcal{S}_t\} \cup \{o_s : s \in \mathcal{S}_t^k\}. \tag{1}$$

For each agent, we choose the smallest k for which adding one more search observation improves pointer coverage by at most one percentage point, subject to the average trajectory containing at least one search observation to evict.

We apply this rule directly to the KV cache rather than deleting search observations from the prompt. Subsequent tokens have already been computed with earlier search observations in context, so their cached keys and values can retain information incorporated from those observations even after the original entries are evicted. In contrast, prompt deletion re-encodes subsequent tokens without the removed search observations. Consistent with this distinction, Search-Selective KV Eviction requires fewer rounds than Search Masking, which removes the same search observations from the prompt (Figure 4a).

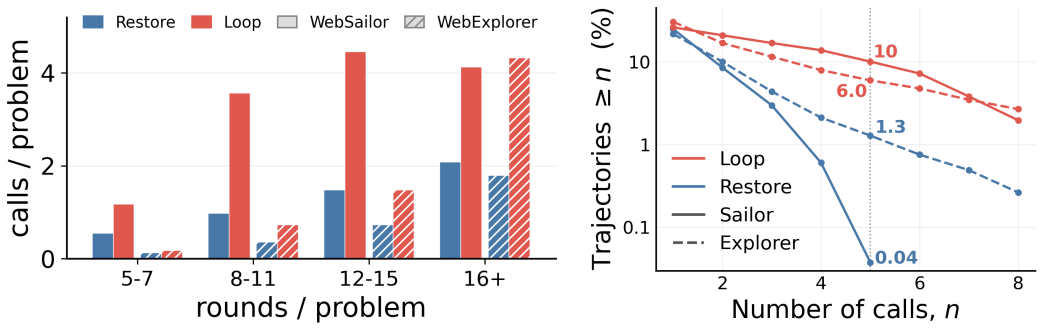
3.2 Part 2: Loop Gating

Removing older search observations reduces KV-cache usage but can increase the number of rounds. Since each additional round incurs tool execution and model inference, KV-cache reduction alone may not reduce overall inference cost. Search-Selective KV Eviction adds fewer rounds than Search Masking, but still increases the round count by approximately 5% on WebSailor and 11% on WebExplorer relative to the uncompressed baseline (Figure 4a). We therefore introduce a second component to limit these additional rounds.

3.2.1 Motivation: Separating Recovery from Repetition

We first examine repeated searches. We define *re-search* as a search whose query is similar to one issued earlier in the trajectory:

$$\max_{q \in Q_t,\ q' \in H_{t-1}} \mathrm{sim}(q, q') \ge \tau, \qquad \mathrm{sim}(q, q') = \max_{T \in \{T_w, T_b\}} \frac{|T(q) \cap T(q')|}{|T(q) \cup T(q')|}, \tag{2}$$



(a) Restore and Loop counts by trajectory length

(b) Tail distribution of Restore and Loop counts

Figure 5: Restore and Loop statistics under Search-Selective KV Eviction. **(a)** Average Restore and Loop counts by trajectory-length bucket. Both increase with trajectory length, making total rounds insufficient to distinguish useful recovery from repetition. **(b)** Fraction of trajectories with at least n Restores or Loops. Restore counts decay rapidly, whereas Loops exhibit a substantially longer tail.

where Q_t contains the current queries, H_{t-1} contains earlier queries, and T_w and T_b extract word tokens and character bigrams from normalized queries. As shown in Figure 4b, re-search accounts for the entire increase in search calls on WebSailor and most of the increase on WebExplorer. However, blocking all re-search could prevent useful recovery of evicted information. We therefore distinguish re-search by what happens next.

Restore. A re-search is a Restore if the next call browses at least one page not already in context, adding a new browse observation that is retained thereafter.

Loop. A re-search is a Loop if the next call is another search, browses only pages already in context.

This distinction separates re-search that expands the retained browse context from repetition that does not, without requiring semantic evaluation of the search result. Moreover, as shown in Figure 4b, Loops account for most of the increase in re-search under Search-Selective KV Eviction on both agents. We therefore limit accumulated Loops while leaving Restores unpenalized.

3.2.2 Proposed Method: Loop Gating

Loop-based budgeting. Longer trajectories contain more Restores as well as more Loops (Figure 5a), so limiting total rounds may interrupt useful recovery together with repetition. Restore counts remain low in most trajectories, whereas Loops accumulate more frequently (Figure 5b). More importantly, Restores expand the retained browse context, whereas Loops do not. We therefore place a budget N on the cumulative Loop count, while Restores do not consume the budget.

Loop classification and gating. The gate first detects re-search using the query-similarity criterion in Eq. 2. Each re-search is executed normally, while its classification as a Restore or Loop is deferred until the next proposed tool call. Before executing that call, the gate increments the Loop count if the re-search is classified as a Loop. Once the Loop count reaches N, tool use terminates and the agent generates its final answer from the accumulated history. The gate uses only tool-call arguments and lightweight trajectory state, including previous queries, opened URLs, and the cumulative Loop count, with no additional model call.

4 Experiments

4.1 Experimental Setup

Agents and Benchmarks. We evaluate two open-source deep research agents: WebExplorer-8B (Liu et al., 2025), served with vLLM (Kwon et al., 2023), and WebSailor-32B (Li et al., 2025), served with SGLang (Zheng et al., 2024). Their browse tools use Qwen2.5-32B-Instruct and Qwen2.5-72B-Instruct (Qwen et al., 2025), respectively, as auxiliary models. Additional implementation details are provided in Appendix C. We use three benchmarks, WebWalkerQA (Wu et al.,

Table 2: Main evaluation results. avg@3 and pass@3 are reported in percent, and $\mathrm{KV}_{\mathrm{avg}}$ and $\mathrm{KV}_{\mathrm{max}}$ are in GB. SWE-style denotes the observation-masking rule from SWE-agent, applied to both search and browse observations.

		WebExplorer-8B					WebSailor-32B				
Method	Level	avg@3	pass@3	Round	KV_{avg}	KV_{max}	avg@3	pass@3	Round	KV_{avg}	KV_{max}
WebWalkerQA											
Base	—	51.23	62.50	8.11	1.47	13.55	52.40	62.50	5.64	2.27	7.62
CPC	prompt	42.11	57.65	12.79	1.26	8.86	41.86	54.41	10.77	2.99	8.21
SWE-style	prompt	52.79	65.29	10.96	0.83	3.34	52.35	63.68	8.38	1.59	6.11
FastKV	cache	46.52	59.12	10.84	0.91	8.05	44.36	61.62	6.41	1.88	5.78
ChunkKV	cache	45.54	59.26	10.99	0.87	10.49	41.86	54.85	6.58	1.92	6.81
SearchTrimmer	cache	51.72	64.41	8.49	0.86	2.34	52.51	65.00	5.76	1.50	5.43
xbench-DeepSearch											
Base	—	62.33	80.00	8.80	2.65	14.99	63.67	76.00	5.28	2.49	7.59
CPC	prompt	60.00	75.00	12.08	1.73	9.66	64.33	73.00	7.56	2.29	8.47
SWE-style	prompt	63.00	80.00	11.28	1.25	4.49	67.67	78.00	6.98	1.78	4.87
FastKV	cache	63.00	80.00	10.33	1.33	7.16	63.67	77.00	6.76	2.24	6.72
ChunkKV	cache	62.67	79.00	10.23	1.36	7.93	63.33	77.00	6.79	2.28	7.07
SearchTrimmer	cache	63.67	77.00	8.53	1.04	3.67	66.81	79.00	5.27	1.70	5.20
GAIA											
Base	—	40.45	59.22	12.33	2.25	14.17	44.98	58.25	6.01	2.15	6.90
CPC	prompt	35.60	51.46	17.66	1.55	7.98	33.98	47.57	9.37	2.17	7.09
SWE-style	prompt	43.37	55.37	16.23	1.06	4.99	43.37	54.37	7.09	1.30	6.18
FastKV	cache	37.22	51.46	15.38	1.37	8.48	39.48	51.46	8.82	2.10	4.58
ChunkKV	cache	38.83	53.40	14.19	1.17	6.78	39.16	56.31	6.64	1.71	4.67
SearchTrimmer	cache	41.42	58.25	11.59	0.96	3.82	46.59	61.17	6.33	1.38	4.92

2025a), xbench-DeepSearch (Chen et al., 2025), and the text-only validation subset of GAIA (Mialon et al., 2024) with 103 questions. An LLM judge scores each answer against the gold answer, DeepSeek-V4.1-Flash (Xu et al., 2026) for WebExplorer-8B and Qwen2.5-72B-Instruct for WebSailor-32B.

Tools. *Search* retrieves URLs and snippets for multiple queries from Google Search through SerpAPI (SerpAPI, 2025). *Browse* fetches page content from the given URLs through Jina Reader (Jina.ai, 2025) and uses an auxiliary model to generate summaries.

Baselines. We compare against four training-free methods from two categories. For static compression of long inputs, CPC (Liskavets et al., 2025) compresses the prompt before encoding, while FastKV (Jo et al., 2026) and ChunkKV (Liu et al., 2026) compress the KV cache. We apply all three to search observations only, matching the compression scope of Search-Selective KV Eviction. For observation-window compression, we use the masking rule from SWE-agent (Yang et al., 2024), extending its single observation stream to a shared window over search and browse observations. Baseline configurations and implementation details are provided in Appendix F.

Hyperparameters. We set $k = 2$ for WebExplorer-8B and $k = 1$ for WebSailor-32B using the selection rule in Section 3.1.2. We use $\tau = 0.7$ for both agents, with $N = 5$ for WebExplorer-8B and $N = 4$ for WebSailor-32B.

Metrics. We report avg@3, the mean accuracy over three rollouts, and pass@3, the fraction of questions answered correctly in at least one rollout. For efficiency, we report rounds per question and average and maximum KV-cache usage over all prefill steps (KV avg and KV max, in GB). KV-cache usage is computed from the number of attended tokens and the per-token KV size.

4.2 Accuracy Evaluation

Table 2 compares SearchTrimmer with the uncompressed agent (Base) and four baselines. SearchTrimmer preserves avg@3 while reducing KV-cache usage. Across WebWalkerQA, xbench-DeepSearch, and GAIA, avg@3 increases by 0.49, 1.34, and 0.97 on WebExplorer-8B, and by 0.11, 3.14, and 1.61 on WebSailor-32B, respectively. Rounds remain within approximately 6% of Base on both agents. Average KV-cache usage decreases by 41–61% on WebExplorer-8B and 32–36% on WebSailor-32B, while maximum usage decreases by 73–83% and 29–31%, respectively.

Table 3: Wall-clock time at fixed concurrency, four workers for WebExplorer-8B and five for WebSailor-32B. S / B denotes the mean number of search and browse calls per question. Round and S / B are means over three rollouts, and Wall-time denotes the elapsed time to complete all questions in a benchmark at the specified concurrency, reported in 10^3 s and measured once per method per benchmark. † denotes SearchTrimmer without Loop Gating.

	WebWalkerQA			xbench-DeepSearch			GAIA		
Method	Round	S / B	Wall-time	Round	S / B	Wall-time	Round	S / B	Wall-time
WebExplorer-8B									
Base	8.11	4.24 / 2.87	27.8	8.80	6.07 / 1.75	5.7	12.33	7.60 / 3.74	10.3
SWE-style	10.96	5.54 / 4.44	37.8	11.28	7.72 / 2.61	6.5	16.23	9.79 / 5.55	13.5
SearchTrimmer†	9.05	4.80 / 3.26	30.2	9.53	6.83 / 1.74	6.0	13.74	8.28 / 4.48	11.5
SearchTrimmer	8.49	4.43 / 3.10	30.0	8.53	5.80 / 1.79	5.1	11.59	6.81 / 3.91	9.7
WebSailor-32B									
Base	5.64	2.17 / 2.47	13.8	5.19	2.11 / 2.09	1.8	6.01	2.50 / 2.40	2.1
SWE-style	8.38	3.80 / 3.58	21.1	6.98	2.89 / 3.07	2.4	7.10	3.21 / 2.61	2.5
SearchTrimmer†	5.92	2.66 / 2.29	13.7	5.21	2.57 / 1.67	1.7	6.25	2.93 / 2.33	2.2
SearchTrimmer	5.53	2.39 / 2.15	12.8	4.94	2.36 / 1.55	1.6	6.19	2.83 / 2.27	2.1

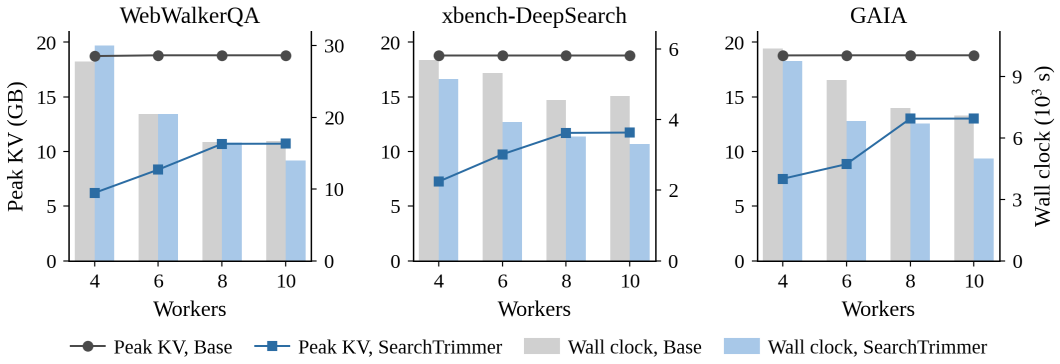


Figure 6: Throughput at a fixed memory budget on WebExplorer-8B. Lines show the peak KV memory in use (left axis) and bars show the time to run the whole benchmark (right axis). Each point is from a single run.

4.3 Efficiency Evaluation

Wall-clock time at fixed concurrency. Table 3 reports one run per method with four workers for WebExplorer-8B on vLLM and five for WebSailor-32B on SGLang, each sharing a single server per agent. The tool-call composition helps explain the timing differences. Recovering a browse observation requires a page fetch and an auxiliary-model call, whereas recovering a search observation requires only another search (Section 3.1.1). SWE-style masking evicts browse observations as well and adds 0.86–1.81 browse calls per question on WebExplorer-8B and 0.21–1.11 on WebSailor-32B relative to Base. Ungated SearchTrimmer adds at most 0.74 on WebExplorer-8B, reduces browse calls on every WebSailor-32B benchmark, and runs 8–20% and 12–35% faster than SWE-style masking. Loop Gating then reduces search calls by 8–18% on WebExplorer-8B and 3–10% on WebSailor-32B relative to ungated SearchTrimmer, and wall-clock time by 1–16% and 4–7%. The timing gain varies across benchmarks, as tool-call counts alone do not determine execution time. Retaining browse observations thus limits the increase in costly browse calls, while Loop Gating further reduces search activity. Together, these effects help translate history compression into lower wall-clock time than SWE-style masking.

Throughput at a fixed memory budget. We serve WebExplorer-8B from one NVIDIA L40S (46 GB) with vLLM at a `gpu_memory_utilization` of 0.8, which leaves 18.8 GB for the KV cache. Figure 6 compares the uncompressed agent and SearchTrimmer as the number of concurrent workers rises from four to ten. The uncompressed agent fills the pool at every worker count, whereas SearchTrimmer holds 33–40% of it at four workers and 57–69% at ten. At four workers, memory

Table 4: Effect of Loop Gating settings. τ is the minimum query similarity for marking re-search, and N is the cumulative Loop budget at which tool use terminates. The first row of each block disables Loop Gating, corresponding to SearchTrimmer† in Table 3. avg@3 is reported in percent.

		WebWalkerQA		xbench-DeepSearch		GAIA	
τ	N	avg@3	Round	avg@3	Round	avg@3	Round
WebExplorer-8B							
no gate		52.11	9.05	64.33	9.53	43.04	13.74
0.7	4	50.64	8.38	63.67	8.63	41.42	11.24
0.7	5	51.72	8.49	63.67	8.53	41.42	11.59
0.8	4	50.49	8.61	62.67	8.71	40.78	12.72
0.8	5	52.16	8.78	62.33	9.25	44.01	12.32
WebSailor-32B							
no gate		52.06	5.92	64.33	5.21	43.37	6.25
0.7	4	52.51	5.76	66.81	5.27	46.59	6.33
0.7	5	51.47	5.91	66.67	5.47	45.47	5.95
0.8	4	51.47	5.53	62.31	5.33	40.78	6.38
0.8	5	51.32	5.73	64.33	5.52	47.90	6.21

is not the bottleneck and the two agents differ by their rounds, so SearchTrimmer is 7% slower on WebWalkerQA and 6–11% faster on the other two. Past eight workers, the uncompressed agent no longer benefits from additional workers. Its wall clock moves from 16.6 to 16.7 (10^3 s) on WebWalkerQA and from 4.6 to 4.7 on xbench-DeepSearch, and falls only 5% on GAIA. SearchTrimmer continues to benefit, since the memory it frees accommodates the added trajectories. At ten workers it is $1.19\times$ to $1.42\times$ faster than the uncompressed agent. Appendix E reports the underlying numbers.

4.4 Ablation: gate settings τ and N

Table 4 varies the query-similarity threshold τ and cumulative Loop budget N, with eviction width k fixed. For WebExplorer-8B, all tested settings reduce rounds relative to the ungated agent. Lowering τ consistently reduces rounds, and lowering N generally provides further reductions, with varying accuracy changes. For WebSailor-32B, lowering N also reduces rounds on WebWalkerQA and xbench-DeepSearch at both thresholds, while GAIA favors the larger budget. These results support using N to control continued exploration, with τ determining which re-searches are eligible for Loop classification. The resulting accuracy–round balance varies across benchmarks.

5 Conclusion

We introduced SearchTrimmer, a training-free framework that jointly addresses growing KV-cache usage and additional rounds induced by history compression in deep research agents. Search-Selective KV Eviction exploits differences among history content types to remove older search observations while retaining browse observations and reasoning. Loop Gating distinguishes re-search that leads to new retained browse evidence from repetition, and limits the cumulative count of the latter. Together, these components reduce retained history and repeated tool use without retraining or an additional model. Across three benchmarks, SearchTrimmer reduces average KV-cache usage by 41–61% on WebExplorer-8B and 32–36% on WebSailor-32B, with maximum KV-cache reductions of 73–83% and 29–31%, respectively, while matching or improving avg@3 relative to the uncompressed agents. Round counts remain within approximately 6% of their baselines. On WebExplorer-8B, the memory savings also enable better scaling with concurrent workers under a fixed KV-cache budget. These results highlight the importance of considering both what an agent retains and how it behaves after information is removed.

AI USE STATEMENT

We used generative AI tools to assist with drafting and revising the manuscript, discussing analysis designs and interpretations, refining pseudocode and LaTeX formatting, and creating preliminary figure designs. The auxiliary models and LLM judges used in the experimental pipeline are described in Section 4.1. The authors are responsible for the final methodology, experimental results, figures, and claims. All AI-assisted material included in the paper was reviewed and revised by the authors.

REPRODUCIBILITY STATEMENT

Section 4.1 lists the agents, benchmarks, tools, judges, and hyperparameters. Appendix A describes the trajectory analysis behind Section 3.1.1, including how facts are extracted and matched and how k is set. Appendix C describes the KV eviction on the paged caches of vLLM and SGLang, and Appendix D gives the full Loop Gating procedure in Algorithm 1. Appendix F gives the configuration of every baseline, and Appendix E reports the numbers behind Figure 6. We will release the code, the agent scaffold, and the evaluation scripts upon publication.

REFERENCES

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, et al. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations. *arXiv preprint arXiv:2506.13651*, 2025.

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.

Weizhong Huang, Jinchao Zhang, and Xiawu Zheng. Commitkv: Lifecycle-aware kv cache compression via commit transitions for multi-turn agents. *arXiv preprint arXiv:2608.07855*, 2026.

Jina.ai. Jina, 2025. URL `https://jina.ai/`.

Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim. Fastkv: Decoupling of context reduction and kv cache compression for prefill-decoding acceleration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32167–32186, 2026.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025.

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.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025.

Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. Acm: Agentic context management for long horizon tasks. *arXiv preprint arXiv:2607.23809*, 2026.

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

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. The complexity trap: Simple observation masking is as efficient as llm summarization for agent context management. *arXiv preprint arXiv:2508.21433*, 2025.

Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane K Luke. Prompt compression with context-aware sentence encoding for fast and improved llm inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24595–24604, 2025.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, et al. Webexplorer: Explore and evolve for training long-horizon web agents. *arXiv preprint arXiv:2509.06501*, 2025.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 38:28728–28778, 2026.

Yijun Lu, Rui Ye, Yuwen Du, Jiajun Wang, Songhua Liu, and Siheng Chen. Longseeker: Elastic context orchestration for long-horizon search agents. *arXiv preprint arXiv:2605.05191*, 2026.

Venkatesha Matam and Keon Kim. Memdecay: Region-aware kv cache eviction for efficient llm agent inference. *arXiv preprint arXiv:2607.10582*, 2026.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

SerpAPI. Serpapi: Google search api, 2025. URL `https://serpapi.com/?gad_source=1&gad_campaignid=1061187028&gbraid=0AAAADD8kqObrG_Yhfov4tkhegHlcAW-v&gclid=CjwKCAjwz5nGBhBBEiwA-W6XRPAgJXyoTwlsU-elg7bW5iIjUA8btM6oK3A_sp2D95exzIyaNjNmPRoCw6cQAvD_BwE`.

Zhengwei Tao, Jialong Wu, Wenbiao Yin, Pu Wu, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, et al. Webshaper: Agentically data synthesizing via information-seeking formalization. In *International Conference on Learning Representations*, 2026.

MindDR Team and Li Auto Inc. Mind deepresearch technical report. *arXiv preprint arXiv:2604.14518*, 2026.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, et al. Webwalker: Benchmarking llms in web traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10290–10305, 2025a.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Robert Tang, et al. Webdancer: Towards autonomous information seeking agency. *Advances in Neural Information Processing Systems*, 2026a.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025b.

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, et al. Contextbudget: Budget-aware context management for long-horizon search agents. *arXiv preprint arXiv:2604.01664*, 2026b.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4.1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

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

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

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

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

A How Facts Are Counted

This section describes the analysis behind Section 3.1.1. We first describe the trajectories, what counts as a fact, and how a fact is matched to a content type. We then describe how each table is built. Table 1 reports the share of tokens and the facts held by each content type at the final round, and Table 6 reports the same per benchmark. Table 7 reports where a fact that passes through search reappears afterwards, and Table 8 reports the facts lost under a window of search observations and under a window of all observations. Table 9 reports how many browses follow one search, and Table 10 reports where the agent finds the URL it browses.

Trajectories. We analyze trajectories of WebExplorer-8B and WebSailor-32B on WebWalkerQA, xbench-DeepSearch, and GAIA, with three iterations per benchmark and 2,649 rollouts per agent. These trajectories are collected without compression and separately from the runs in Section 4. Where two numbers are given, the first refers to WebExplorer-8B and the second to WebSailor-32B.

Correct rollouts. We judge correctness as in Section 4, which gives 1,455 and 1,398 correct rollouts. We keep the rollouts that make at least one tool call before the answer, which leaves 1,455 and 1,397.

Facts. We define a fact as a name, a date, or a number from the gold answer that appears in a correct rollout outside the question and before the final answer. One gold answer may yield several facts, so a rollout can contribute more than one. A fact may be held by more than one content type, and a content type holds a fact alone if no other type holds it. Of the 1,455 and 1,397 rollouts, 668 and 658 hold at least one fact, giving 1,116 and 1,122 facts (Table 5). The remaining rollouts yield no string specific enough to extract and are excluded.

Table 5: Rollouts and facts in the analysis.

	WebExplorer-8B	WebSailor-32B
Rollouts	2,649	2,649
Correct	1,455	1,398
With a tool call	1,455	1,397
With a name fact	410	427
With a date or number fact	333	316
With any fact	668	658
Facts	1,116	1,122
Facts through search with a later round	957	952

Extraction. The gold answer is split on clause punctuation, and from each clause we keep quoted titles, English proper names, and CJK strings that read as organizations or alphanumeric names. A connector such as *of* or *de* may join an English name, whereas *and* does not, so two names joined by *and* become two facts. English names shorter than eight normalized characters and common nouns are dropped. Dates and numbers come from a separate pattern that recognizes ISO, English, and CJK dates, times, percentages, quantities with unit words, years, and thousands-separated numbers. When candidates of either kind overlap, only the longest is kept. A candidate that the name rules pick up but that is entirely a date or number is counted as a date or number. Dates and numbers have no minimum length, so a short one can match by chance in a content type that does not hold the answer. Such a fact is then held by more than one content type and does not enter the sole count.

Matching. A content type holds a fact if the fact appears in its text. We normalize both the fact and the text in the same way before matching. Case and punctuation are removed, and CJK numerals and full-width punctuation are mapped to their ASCII forms. Dates are reduced to year, month, and day regardless of format, and thousands separators are ignored.

Share, sole, and density. For each fact we record which content types hold it in the prompt of the final round. Share is the fraction of tokens across the three types, sole is the fraction of facts held by

that type only, and density is sole divided by share. Share is measured over the rollouts that hold at least one fact, with the fixed prefix of system prompt and question left out.

By benchmark. Table 6 reports Table 1 per benchmark. Browse is the densest content type on every benchmark and for both agents. Whether reasoning or search is denser depends on the agent. Reasoning is denser than search on every benchmark for WebExplorer-8B, and sparser or equal for WebSailor-32B. On GAIA, reasoning holds a larger share of the facts alone for WebExplorer-8B than on the other benchmarks, as several questions are answered by a computation in a thought rather than by a string retrieved from the web.

Table 6: Share, sole, and density by benchmark and content type. Share is the percentage of tokens across the three types, sole is the percentage of facts held by that type only, and parentheses give the number of facts in each benchmark.

		Search			Browse			Reasoning		
Agent	Benchmark	Share	Sole	Density	Share	Sole	Density	Share	Sole	Density
WebExplorer-8B	WebWalkerQA (1,042)	88.0	4.2	0.048	3.5	5.8	1.63	8.5	0.7	0.079
	xbench-DS (27)	91.6	14.8	0.162	1.0	22.2	21.9	7.4	3.7	0.50
	GAIA (47)	89.0	0.0	0.00	2.8	10.6	3.79	8.2	8.5	1.04
	All (1,116)	88.4	4.3	0.049	3.3	6.4	1.93	8.4	1.1	0.129
WebSailor-32B	WebWalkerQA (1,053)	71.3	1.6	0.023	11.2	14.2	1.27	17.6	0.1	0.005
	xbench-DS (22)	77.0	9.1	0.118	9.4	13.6	1.45	13.6	0.0	0.00
	GAIA (47)	69.0	0.0	0.00	12.7	21.3	1.67	18.3	0.0	0.00
	All (1,122)	71.3	1.7	0.024	11.2	14.4	1.29	17.4	0.1	0.005

Copy rate. A fact passes through search if a search observation holds it. For Table 7, we keep the facts followed by at least one more round before the answer, so that a copy has a place to appear. This leaves 957 and 952 facts. For each fact, we look for a copy in a thought or browse observation within the following one or two rounds. A copy appears in the next round for 85.8% and 77.7% of these facts, and within two rounds for 91.1% and 92.3%.

Table 7: Facts that pass through search and where a copy appears afterwards. A copy is counted in a thought or browse observation only. Rates are over the facts followed by at least one more round.

	WebExplorer-8B	WebSailor-32B
Facts through search with a later round	957	952
With a copy in the next round	85.8%	77.7%
With a copy within two rounds	91.1%	92.3%

Loss under a window. Table 8 reports whether a window loses any fact at the final round. We keep the k most recent search observations, remove every older one, leave browse observations and reasoning intact, and count the facts that no content type holds any more. For comparison, we apply the same window to all observations, keeping the w most recent observations of either kind and removing the rest. The search window loses none of the 1,116 and 1,122 facts at $k = 1$, and wider windows lose none either. The observation window loses 41 and 91 at $w = 1$, 16 and 22 at $w = 2$, and 3 and 7 at $w = 3$. The observation window removes browse observations along with search observations, so a fact held only by browse is lost with them.

Runs of browses. We define a run as the browses between a search and the next search, and count only the searches followed by at least one browse. Table 9 reports the distribution of run length and the share of trajectories with at least one run of a given length. An observation window of width w evicts a search observation once w browses have followed it. A run longer than w therefore continues from a search observation the window has already removed, and the share of such runs is the share of searches from which the agent is still choosing pages after eviction.

Table 8: Facts lost at the final round under a window of search observations, which keeps the k most recent search observations, and under a window of all observations, which keeps the w most recent observations.

	WebExplorer-8B			WebSailor-32B		
Window size	1	2	3	1	2	3
Search observations kept (k)	0	0	0	0	0	0
Observations kept (w)	41	16	3	91	22	7

Table 9: Browses that follow one search. A run is the browses between a search and the next search. The upper rows report shares of runs by length and the mean run length. The lower rows report the share of trajectories with at least one run of the given length.

	WebExplorer-8B	WebSailor-32B
Runs of length 1	78.3%	59.1%
Runs of length 2	17.6%	27.0%
Runs of length 3 or more	4.1%	13.9%
Mean run length	1.27	1.66
Trajectories with a run of 2 or more	39.0%	47.6%
Trajectories with a run of 3 or more	8.6%	17.7%

Pointer tracing. For Table 10, we trace every browse to where the agent found the URL. Each URL is counted only the first time it is browsed. We exclude a URL that a search observation lists and that a page browsed earlier also contains, since the trajectory does not show which of the two the agent used. For the remaining browses, we record which search observation listed the URL, counting back from the most recent one at the time of the browse.

Table 10: Where the agent finds the URL it browses. The first column reports the share of browses whose URL a search observation lists. The other columns report the share of those browses whose URL is in the k most recent search observations at the time of the browse.

	From a search observation	Within the k most recent		
		$k = 1$	$k = 2$	$k = 3$
WebExplorer-8B	95.5%	95.4%	98.8%	99.5%
WebSailor-32B	85.9%	93.9%	98.5%	99.5%

Setting k. We set k per agent from the uncompressed trajectories using the rule in Section 3.1.2. Pointer coverage is the fraction of first browses whose URL appears in one of the k most recent search observations (Table 10). Increasing k from 1 to 2 raises coverage by 3.4 and 4.6 points on WebExplorer-8B and WebSailor-32B, whereas increasing k from 2 to 3 adds only 0.7 and 1.0. The coverage criterion therefore selects $k = 2$ for both agents. Whether this k leaves search observations to evict depends on how often each agent searches. WebExplorer-8B issues 4.1–8.1 searches per trajectory across the three benchmarks, so $k = 2$ still evicts search observations in the average trajectory. WebSailor-32B issues only 2.1–2.3, so $k = 2$ would leave the average trajectory with nothing to evict. We therefore set $k = 1$ for WebSailor-32B, at a coverage of 93.9%. This lower coverage is one source of the additional rounds discussed in Section 3.2.

B What a Thought Records

Section 3.1.1 retains reasoning in full and shows that facts in a search observation reappear in a later thought or browse observation. This section shows what a thought records at its round and how a later round draws on it. For each, we give an excerpt from an uncompressed trajectory and a count over the trajectories.

The state at a round. The following excerpt, from a GAIA trajectory of the uncompressed WebExplorer-8B agent, shows the question, the first search, and the thought that follows it. The first sentence of the thought records what the search settled, the species behind the museum object the question names. The second sentence records what the question still asks and the search has not yet answered.

> **Question.** The object in the British Museum's collection with a museum number of 2012,5015.17 is the shell of a particular mollusk species. According to the abstract of a research article published in Science Advances in 2021, beads made from the shells of this species were found that are at least how many thousands of years old?
>
> **Search.** "British Museum collection 2012,5015.17 mollusk shell"; "British Museum object number 2012,5015.17"
>
> **Thought.** Great! I found that the British Museum object 2012,5015.17 is a shell of the Mediterranean species 'Nassa gibbosula'. Now I need to search for research articles published in Science Advances in 2021 that mention beads made from shells of this species and their age.

We count how often a thought records each part, what the search settled and what remains open, by scanning the thoughts of the uncompressed WebExplorer-8B agent on GAIA for marker phrases. The scan covers 1,249 thoughts over 103 questions in the first iteration. A settled marker is a phrase such as "I found", "we know", "confirmed that", or "the answer is", and an open marker is a phrase such as "I need", "need to find", "still missing", or "what remains". A settled marker appears in 28% of thoughts and an open marker in 37%, with 9% carrying both. The shares are stable across the three iterations, settled 27–28%, open 37–39%, and both 9–12%. In the first iteration, the remaining 44% state only the next action.

The count is a phrase match, not a reading of each thought. A phrase outside the list, such as "Now let me calculate", is not counted as an open marker, so the shares are lower bounds on how often a thought records each part.

What is carried forward. The following excerpt, from a WebWalkerQA trajectory of the same agent, shows a thought and the answer written seven rounds later. The question asks for the candidates for VP of Accessibility in the SIGCHI 2024 elections and the membership requirement for voting. The search of round 1 lists the candidates, and the thought of round 2 records them. Rounds 3 to 8 search and browse membership and bylaws pages for the voting requirement, and none of those observations names a candidate. The answer of round 9 takes the candidates from the thought of round 2.

> **Round 2 (thought).** Great! I found the information about the VP of Accessibility candidates. From the search results, I can see that there are two candidates: 1. Dragan Ahmetovic 2. Fernando Loizides. Now I need to find more specific information about the voting requirements.
>
> **Rounds 3 to 8.** *Two searches and four browses on membership and voting eligibility. No observation names a candidate.*
>
> **Round 9 (answer).** There are two candidates for VP of Accessibility in the SIGCHI 2024 elections: 1. Dragan Ahmetovic 2. Fernando Loizides. [. . .] active SIGCHI membership.

A result that the answer takes from a thought two or more rounds earlier, with no observation repeating it in between, occurs in 6.9% and 4.1% of the correct trajectories of WebExplorer-8B and WebSailor-32B. The case also shows why reasoning has to stay. Under Search-Selective KV Eviction with $k = 2$, the search of round 1 is evicted before round 7, and from then on the thought of round 2 is the only copy of the candidates in the context. If the thought were evicted as well, rounds 7 to 9 would run with no copy at all, and the answer could not name the candidates without a new search.

C Implementation on Paged KV Caches

This section describes the implementations in vLLM for WebExplorer-8B and SGLang for WebSailor-32B. Both retain search observation text in the prompt and mark older search observations for eviction inside the engine.

In vLLM, each round is a new request, and the prefix cache reuses pages from earlier rounds. The agent supplies the token ranges to evict, which the engine handles during cache lookup and attention.

Marking. With each request, the agent attaches the token ranges of every search observation older than the k most recent. A page of $B = 16$ tokens is marked as search only when every one of its tokens lies inside the range. The pages at the two ends of a search observation can hold a few reasoning tokens as well, so they are not marked.

Lookup. When lookup reaches a page marked as search, the engine inserts a placeholder. The placeholder is treated as already computed and does not reference a physical KV page, so the search tokens are not prefilled and their KV entries are not held by the request. The prompt text is unchanged, so the hashes of the later pages are the same as in the previous round. The reasoning and browse pages after the evicted search are reused as before. The request thus reads the prefix, every reasoning and browse page, and the k most recent search observations, which is $\mathcal{K}_t$ of Equation. 1.

Attention. Before each forward pass, the placeholders are dropped from the page list, so attention reads exactly the pages of $\mathcal{K}_t$. The remaining tokens keep their positions, so the rotary embedding of every token is the same as in the uncompressed run. The reasoning and browse pages after an evicted search are not recomputed.

SGLang. WebSailor-32B uses the same keep-prompt rule on a streaming session in SGLang: the search text stays in the prompt, and the agent marks the token ranges of every search observation older than the k most recent. We set the page size to $B = 1$, so each token is its own page and a search observation is marked exactly. There are no mixed pages at the two ends, and every marked token is dropped from the physical KV row. The remaining tokens keep their positions, so the rotary embedding matches the uncompressed run, and attention reads exactly $\mathcal{K}_t$. Only the new suffix is prefilled.

Algorithm 1 Loop Gating

Require: Similarity threshold $\tau \in (0, 1]$, integer Loop budget $N \geq 1$
$\mathcal{H} \leftarrow \emptyset, \mathcal{U} \leftarrow \emptyset$ ▷ Queries posed and URLs opened
$p \leftarrow 0, L \leftarrow 0$ ▷ Pending re-search and cumulative Loops
for each proposed search or browse call a **do**
  if $p = 1$ **then**
    if $a = \mathrm{Browse}(V)$ and $V \setminus \mathcal{U} \neq \emptyset$ **then**
      $L \leftarrow L$ ▷ Restore
    else
      $L \leftarrow L + 1$ ▷ Loop
    end if
  end if
  $p \leftarrow 0$
  if $L \geq N$ **then**
    return FINALANSWERFROMHISTORY ▷ Terminate tool use without executing a
  end if
  if $a = \mathrm{Search}(Q)$ **then**
    $s \leftarrow \max_{q \in Q,\, q' \in \mathcal{H}} \mathrm{sim}(q, q')$ ▷ Maximum over an empty set is 0
    Execute a
    $p \leftarrow \mathbb{1}[s \geq \tau]$
    $\mathcal{H} \leftarrow \mathcal{H} \cup Q$
  else if $a = \mathrm{Browse}(V)$ **then**
    Execute a
    $\mathcal{U} \leftarrow \mathcal{U} \cup V$
  end if
end for
$L \leftarrow L + p$ ▷ Count unresolved re-search as Loop for reporting

D LOOP GATE: PROCEDURE AND BUDGET ANALYSIS

D.1 GATE PROCEDURE

Algorithm 1 maintains the query history $\mathcal{H}$, opened URLs $\mathcal{U}$, a pending re-search indicator p, and the cumulative Loop count L. These persist across turns and reset only for a new trajectory.

A detected re-search executes normally. Before executing the next proposed call, the gate resolves the pending re-search: a visit containing an unopened URL is Restore; another search or a visit containing only previously opened URLs is Loop. Only Loop increments L. When L reaches N, tool use terminates before the proposed call executes, and the agent generates its final answer from the accumulated history. A re-search still pending at natural termination is counted as Loop for reporting purposes. Detection and classification use only tool-call arguments and the maintained state, requiring no additional model call.

D.2 ANALYSIS OF THE LOOP BUDGET

We examine budgets $N \in \{4, 5\}$ using ungated trajectories under Search-Selective KV Eviction. For each budget, we select trajectories that would trigger the gate and place the stopping boundary immediately before the call that would resolve the N-th Loop. We measure the fraction of Restore events completed before this boundary and the mean numbers of Loops and Restores remaining afterward.

Most Restore events occur before either stopping boundary, while subsequent re-search consists predominantly of Loops (Table 11). At $N = 5$, 95.8% of Restores on WebSailor and 89.4% on WebExplorer have already occurred. Beyond this point, trajectories contain an average of 1.49 additional Loops versus 0.15 Restores on WebSailor, and 3.87 versus 0.74 on WebExplorer.

These results support using accumulated Loops as a stopping signal after most observed recovery has taken place. They do not imply that all subsequent calls are unnecessary. Section 4.4 evaluates

Table 11: Recovery before and repetition after the stopping boundary in ungated trajectories that would trigger the gate. Before reports the percentage of Restore events completed before stopping. After reports mean additional Loop/Restore counts per selected trajectory.

	WebSailor		WebExplorer	
N	Before (%)	After (Loop/Restore)	Before (%)	After (Loop/Restore)
4	91.9	1.92 / 0.23	83.8	3.74 / 0.87
5	95.8	1.49 / 0.15	89.4	3.87 / 0.74

the resulting accuracy–round trade-off. The trajectory sets are selected separately for each N, so comparisons between budgets do not represent changes within a fixed cohort.

E Throughput Measurements

Table 12 reports the numbers behind Figure 6. Peak is the largest KV memory the engine reports in use over the run, Avg is the mean KV memory in use, and Wall is the time to run the full benchmark.

Table 12: Peak and average KV memory (GB) and wall-clock time (10^3 s) on WebExplorer-8B by number of concurrent workers. Every run shares one KV pool of 18.8 GB. Each entry is from a single run.

		WebWalkerQA			xbench-DeepSearch			GAIA		
Method	Workers	Peak	Avg	Wall	Peak	Avg	Wall	Peak	Avg	Wall
Base	4	18.73	5.20	27.8	18.80	8.97	5.7	18.79	8.29	10.3
	6	18.80	6.85	20.5	18.80	11.44	5.3	18.80	11.16	8.8
	8	18.80	9.68	16.6	18.80	13.00	4.6	18.80	10.37	7.5
	10	18.80	11.85	16.7	18.80	15.39	4.7	18.80	14.44	7.1
SearchTrimmer	4	6.21	2.68	30.0	7.24	3.50	5.1	7.50	2.96	9.7
	6	8.34	3.66	20.5	9.74	4.36	3.9	8.84	3.87	6.8
	8	10.70	4.76	16.5	11.70	5.96	3.5	13.00	5.05	6.7
	10	10.71	5.64	14.0	11.73	6.54	3.3	13.01	5.69	5.0

F Baseline and Hyperparameter Settings

Observation Masking. The SWE-agent rule keeps the w most recent observations and replaces every older observation with a one-line placeholder in the prompt. We count w over search and browse observations together. We set w so that the average KV-cache usage approximately matches that of SearchTrimmer at each agent's chosen k, which gives $w = 3$ for WebExplorer-8B and $w = 2$ for WebSailor-32B.

Static compressors. We apply CPC, FastKV, and ChunkKV to search observations only. The system prompt, question, reasoning, and browse observations are left uncompressed, matching the scope of SearchTrimmer.

CPC removes sentences from each search observation before encoding. A search observation holds one result block per query, and we apply CPC to the body of each block, with the block's search query as the question. CPC ranks the sentences of the body by cosine similarity to the query with a context-aware sentence encoder, a LoRA-tuned Llama-3.2-1B with bidirectional attention, and keeps the top-ranked sentences in their original order up to a keep rate of 0.3 of the block. Each search observation is compressed once, when it enters the history, and remains compressed thereafter.

FastKV and ChunkKV compress the KV cache with a retention rate of 0.35. For both methods, each round prefills and compresses only the newly appended search observation and leaves the KV entries of earlier rounds unchanged. For FastKV, we use a query window of 8 tokens, max-pooling kernel

of 7, and per-KV-head selection of the official implementation. Its token-selective propagation is applied at layer 17 of the 36-layer model, with the propagation rate raised from the default 0.2 to 0.35 so that it retains no fewer tokens than the retention rate. For ChunkKV, we use chunks of 10 tokens as in the original paper and select chunks with SnapKV-style window attention over the last 32 tokens. The implementation pools the attention scores with an average-pooling kernel of 5 before summing them per chunk, selects per KV head, and always keeps the last partial chunk and the window.