ACM: Agentic Context Management for Long Horizon Tasks

Anonymous ACL submission

Abstract

Agentic tasks are inherently long-horizon and multi-turn, constantly accumulating context through interactions with the environment. Existing context compression methods inevitably incur information loss and are triggered by rigid heuristic rules, leaving them misaligned with the agent's evolving reasoning focus. We propose **Agentic Context Management** (ACM), a framework that equips agents with purpose-built context editing tools for lossless context management. Inspired by the interaction between short-term and long-term human memory, the agent autonomously decides when to compress its context, offloads discarded content to an external memory system, and queries it on demand for later retrieval. Building on this framework, we further develop a post-training pipeline that constructs high-quality demonstrations of context management and improves model performance on both search and coding tasks. Further analysis reveals that effective context management reduces peak token pressure, enables extended explorations, and yields more consistent solutions across independent trials.

1 Introduction

Agentic tasks have emerged as a central challenge for LLM-powered autonomous agents (Xu et al., 2024; Xie et al., 2024; Deng et al., 2025; Yang et al., 2024; Wang et al., 2024b; Anthropic, 2024; OpenAI, 2025a). These tasks require agents to formulate adaptive plans, invoke tools (Schick et al., 2023), and adjust their actions in response to environmental feedback. **However, the traces produced by long-horizon agentic tasks are inherently verbose and noisy**. In real-world environments, lengthy tool outputs are often interleaved with failed attempts and redundant observations. When combined with the agent's own reasoning traces, they accumulate into histories that exceed an agent's effective context capacity, even when the underlying model supports nominal context windows of millions of tokens (Gemini Team, Google, 2024; Anthropic, 2025; OpenAI, 2025b).

Prior work has explored several directions to mitigate this limitation. Long-context pretraining extends the window but exhibits measurable degradation (Liu et al., 2024; Hsieh et al., 2024; Bai et al., 2024; Hong et al., 2025). Hybrid attention reduces the cost of processing long inputs but still remains fundamentally bounded by the context window (Dao and Gu, 2024; Liu et al., 2025; Lenz et al., 2025). Context-compression pipelines — which truncate, summarize, or re-render histories into denser formats — represent a promising direction (Wei et al., 2025; Kang et al., 2025). However, these approaches control compression timing through forced, hand-crafted external monitors, relying on heuristic rules that are not well aligned with the model's own reasoning process.

In this paper, we propose a framework that enables model-intrinsic and lossless context management. Specifically, by equipping the agent with a set of well-designed memory tools, we allow the agent itself to decide **when** and **how** to manage its context: identifying irrelevant information, summarizing and offloading it to external memory, and querying it on demand. Our design draws inspiration from the separation between short-term and long-term memory in human cognition (Atkinson and Shiffrin, 1968; Packer et al., 2023) — in-context messages serve as a compact working memory buffer focused on reasoning, while an external store acts as long-term memory ready for future retrieval. This mechanism enables the agent to expand or contract its effective context as its understanding of task progress evolves.

Building on this framework, we further develop an efficient post-training pipeline to help the model internalize context management ability. We adopt a teacher–student framework (Hinton et al., 2015) with dual constraints. In one direction, the stu-

dent performs rollouts *without* context management, and the teacher reviews the resulting trajectories to identify where context management should be inserted, e.g., when the model is stuck in a dead-end loop. In the other direction, the student generates another set of rollouts *with* full access to context management tools, and the teacher identifies where the context management should *not* have been called—replacing it with either a commitment to an answer or a deeper search action. The dual constraints teach the student agent the accurate timing of context management. Using high-quality demonstrations from this dual-constraint pipeline, we improve searching and coding capabilities of Qwen3.5-9B by 29%, 16%, and 8% on BrowseComp-Plus, DeepSearchQA, and SWE-Bench Verified respectively, nearly matching open-source models that are 40× larger in parameters. Analysis reveals that ACM reduces peak token usage by around 20%, increases tool call frequency, and extends the test-time exploration turns. These gains translate into more consistent solutions across independent trials. In summary, our contributions are as follows:

- We introduce *agentic context management*, a paradigm in which the agent autonomously decides **when** and **how** to manage its own context.
- We propose an efficient post-training pipeline that internalizes context management ability into the model itself.
- We demonstrate that effective context management reduces peak token pressure, extends test-time exploration turns, and improves solution consistency across independent trials.

2 Related Works

2.1 Heuristic Context Compression

Heuristic Context Compression reduces context length through external compression modules or hand-designed discarding rules that operate outside the agent's decision process. ReSum (Wu et al., 2025) is among the first frameworks to adopt fixed-timing compression for agentic search tasks. ACON (Kang et al., 2025) trains a compressor that replaces prior histories with a condensed summary, an approach also adopted in Claude's Automatic Context Compression (Anthropic, 2024). COMPASS (Wan et al., 2025) delegates compression to a Meta-Thinker agent that supplies compact contexts to the main agent. DeepSeek-V3.2 (Liu et al., 2025) studies manually designed strategies such as full-history summarization and fixed-ratio truncation, and shows that accuracy continues to improve with step count. MemAgent (Yu et al., 2025) processes inputs chunk by chunk, discarding earlier content while retaining only last-round memory and the question. SideQuest (Kariyappa and Suh, 2026) takes an orthogonal infrastructure-level approach by managing the KV cache directly during long-horizon agentic reasoning. Lu et al. (2025) (SUPO) and Sun et al. (2025) both use reinforcement learning (Shao et al., 2024) to teach the agent to summarize or fold context. Collectively, these approaches demonstrate the utility of context reduction, but they rely on external modules or fixed heuristics rather than on decisions made by the agent during reasoning. Our work instead treats context management as an explicit agent action during task execution. AgentFold (Ye et al., 2025) is similar to our work, but it does not provide a systematic data pipeline for context management. We complement this by open-sourcing a complete data generation pipeline that enables efficient post-training without heavy reinforcement learning.

Method	Compact	Trainable	Lossless	Agent-init
ACE (Zhang et al., 2026)	✗	✗	✗	✗
Mem1 (Zhou et al., 2025)	✗	✓	✗	✗
ReSum (Wu et al., 2025)	✓	✓	✗	✗
ACON (Kang et al., 2025)	✓	✗	✗	✗
SUPO (Lu et al., 2025)	✓	✓	✗	✗
AgentFold (Ye et al., 2025)	✓	✓	✗	✓
ACM (Ours)	✓	✓	✓	✓

Table 1: Comparison of context management approaches. **Compact**: actively compresses working context. **Trainable**: the management policy is learned in training. **Lossless**: raw content is preserved for later retrieval. **Agent-init**: compression is triggered by the agent itself. More details please refer to Appendix A

2.2 Memory-Augmented Context Evolution

Memory-Augmented approaches (Singh et al., 2025; Packer et al., 2023; Park et al., 2023; Wang et al., 2024a; Xu et al., 2025; Fang et al., 2025) maintain an external memory that is updated continuously—typically through reflection, distillation, or optimization—so that the working context remains concise while accumulated knowledge is stored elsewhere. MIPRO (Opsahl-Ong et al., 2024) jointly optimizes instructions and demonstra-

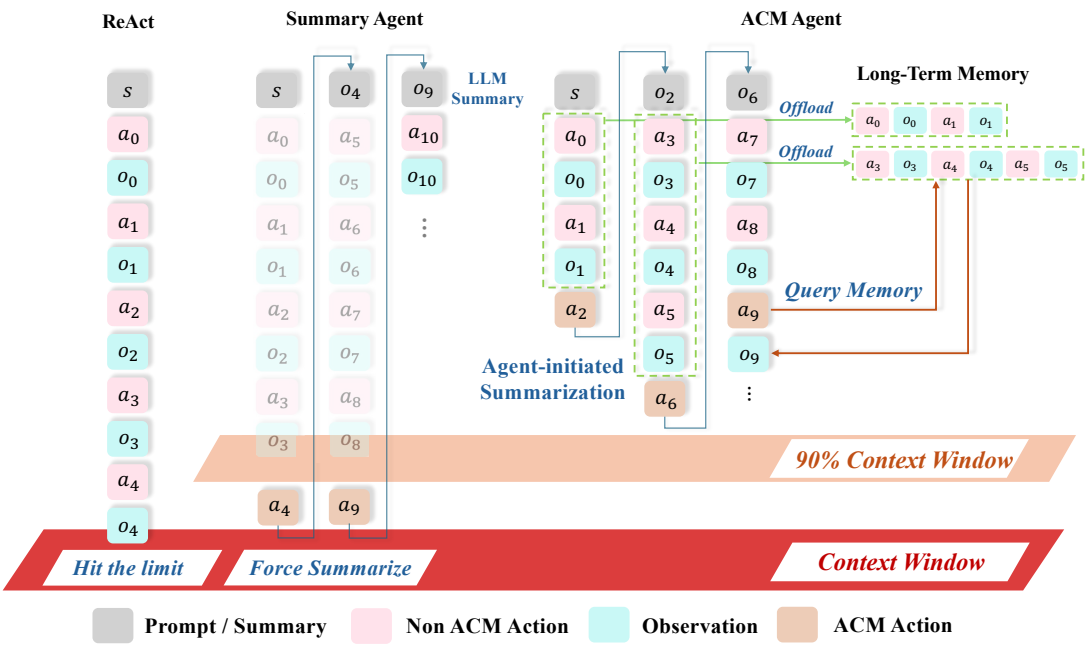


Figure 1: Overview of our ACM Framework. ReAct eventually hits the context limit, while the Summary Agent is forced to compress whenever usage exceeds a predefined threshold (e.g., 90% of the context in the figure) and discards the original messages. The ACM agent autonomously decides *when* to manage its context losslessly.

tions across multi-stage LM programs. Dynamic Cheatsheet (Suzgun et al., 2026) learns a reusable note at test time that records useful strategies. GEPA (Agrawal et al., 2026) evolves prompts and contexts through reflective Pareto search. Agentic Context Engineering (Zhang et al., 2026) treats the context itself as an evolving artifact refined through self-improvement loops. Mem1 (Zhou et al., 2025) consolidates trajectories into a compact internal memory state that is updated each turn. These approaches accumulate knowledge *across tasks* but do not compress the working context within a single episode. ACM, by contrast, is a purely *per-question* method that dynamically compresses and retrieves context through explicit tool calls during reasoning. Table 1 summarizes the key differences.

3 Agentic Context Management Framework

Formulation Let s denote the system prompt, a_t an agent action (reasoning content plus tool calls), and o_t the corresponding environment response at turn t. The agent π_θ conditions on the accumulated history $H_t = \{s, (a_1, o_1), \ldots, (a_{t-1}, o_{t-1})\}$ to produce:

$$a_t \sim \pi_\theta(\cdot \mid H_t), \qquad o_t \sim \pi_\gamma(\cdot \mid a_t, H_t),$$

where π_γ is the model that returns environment response via tool results. The interaction terminates when the agent selects the `finish` action a_T or its context window reaches the limit.

Summary Agent. In the summary-agent paradigm, an external monitor triggers the compression action a_{sum} when context usage exceeds a predefined threshold. The environment returns a summary $o_{\text{sum}} \sim \pi_\gamma(\cdot \mid a_{\text{sum}}, H_t)$, then the agent discards all prior messages and continues reasoning over the updated history $H' = \{s, o_{\text{sum}}\}$.

ACM Agent. We draw inspiration from the interaction between short-term and long-term memory in human cognition: people keep immediately relevant information in working memory while offloading less immediate details into external persistent records, and retrieve them later when the task requires. We introduce two context management tools to enable the agent to mimic the human memory mechanism: `manage_context`, which compresses previous turns into a concise summary and offloads the raw messages to an external file on disk; and `query_memory`, which allows the agent to query the stored raw messages to retrieve information precisely.

The overall mechanism of ACM, along with a comparison to the ReAct and the Summary agent, is illustrated in Figure 1. When the agent decides to manage its context, it invokes `manage_context`

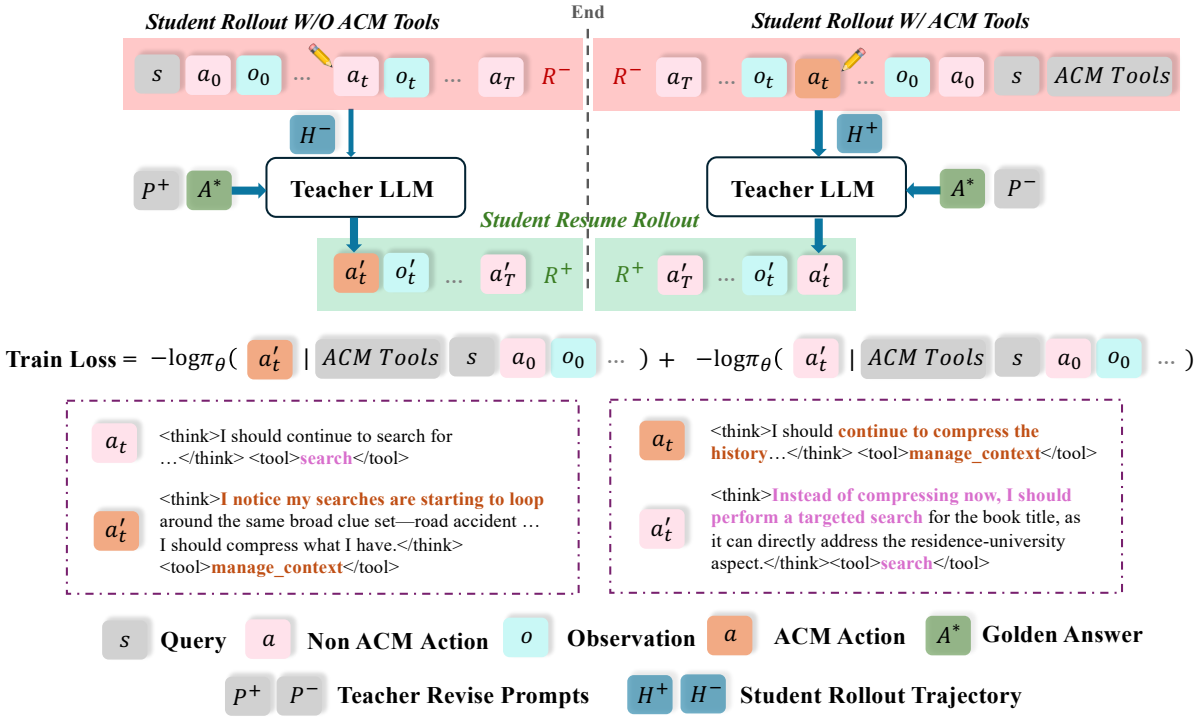


Figure 2: Overview of the dual-constraint training data generation pipeline. A student agent completes task rollouts both with and without context management tools. A teacher model reviews each trajectory against the reference answer and either injects ACM action or replaces non ACM actions. Rejection sampling retains only trajectories where the teacher's interventions lead to successful task completion.

(action a_2, a_6 in Figure 1) to compress all messages up to the previous summary boundary using a summarizer LLM. Crucially, the original messages are not discarded but saved to the agent's external workspace. Each summary is assigned a unique identifier that maps the summary to the corresponding raw messages in external memory. When the agent needs to revisit earlier content, it invokes `query_memory` (action a_9 in Figure 1) with a specified identifier. A querier LLM receives the query along with the raw messages mapped by that identifier, then returns the information related to the query as a tool result.

ACM has two key properties that distinguish it from prior summary-based agents. **1) Information compression is lossless**: all compressed messages are preserved in external storage and are available for the agent to revisit at any time, while the working context stays short and clean. **2) Context management is agent-initiated**: the agent can invoke compression at any point during reasoning process, rather than relying on a fixed schedule or an external trigger. This enables context management to follow the agent's evolving reasoning state and task progress. By allowing compression at any point before the history reaches peak length, the design also alleviates peak token usage pressure.

4 Training Data Generation

As discussed in Section 5.3 and supported by Ye et al. (2025); Lu et al. (2025), even frontier models struggle to determine the appropriate timing for context management. To address this gap, we design a teacher-guided data generation pipeline with dual constraints that is both easy to scale and capable of producing high-quality management demonstrations. We reuse the formulation in Section 3.

4.1 Teacher-Guided Annotation

Our pipeline employs a teacher–student framework with *dual constraints* and operates in two phases, as illustrated in Figure 2.

Phase 1: Student Rollout. A student model completes the task under two conditions—*with* and *without* access to context management tools—producing trajectories denoted H^+ and H^-, respectively. H^+ captures the student's untrained usage behavior of the context management tools, while H^- reflects its ordinary exploration behavior. In Figure 2, the H^- rollout is shown on the left, where the student starts from the system prompt s

alone without ACM tools, whereas the H^+ rollout is shown on the right, starting from s together with the ACM tools.

Phase 2: Teacher Annotation. A *teacher* model receives one of two guided instruction prompts, P^+ or P^-: 1) teacher using P^+ demonstrates *when to use* the context management tools while 2) teacher using P^- demonstrates *when not to* call them. The teacher is additionally provided with the corresponding student trajectory (H^+ or H^-) and the reference answer A^*. It then reviews the trajectory and produces annotations under two complementary constraints:

- **Injection on** H^- (where to *add* context management). Given P^+, the teacher identifies turns at which context management would be beneficial—specifically, points where the student begins querying redundant topics, enters unproductive loops, or has accumulated sufficient context to warrant compression. At each such turn t, the teacher uses a context management tool call a'_t accompanied by a reasoning trace that justifies the compression.

- **Refinement on** H^+ (where to *remove* context management). Given P^-, the teacher identifies turns at which the student's context management calls are premature or unnecessary. At each such turn t, the student has typically either gathered sufficient information but failed to synthesize a final answer, or overlooked key evidence in the retrieved documents that warrants deeper exploration. The teacher replaces the inappropriate context management call a_t with a more productive action a'_t—such as searching for additional evidence, opening a relevant document, or committing to an answer—accompanied by a reasoning trace.

In both cases, the student's original action a_t is replaced with the teacher-annotated action a'_t, and the student rollout resumes from a'_t. Then we train the student based on the negative log-likelihood loss

$$\mathcal{L}_{\text{ACM}}(\theta) = -\,\mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{t\in\mathcal{T}_a(\tau)} \log \pi_\theta(\tilde{a}_t \mid s,\, h_{<t})\right],$$

where $\mathcal{D}$ is the teacher-annotated dataset that pools the refined H^+ and the injected H^- trajectories, $h_{<t} = (a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ is the interleaved history of past actions and tool observations, and $\mathcal{T}_a(\tau)$ restricts the loss to assistant tokens so that system-prompt and tool-output tokens are masked out. The supervised target $\tilde{a}_t$ equals the teacher's correction a'_t at any step where the teacher intervened and the student's original action a_t elsewhere; under this objective the student is jointly trained to invoke context management at productive turns and to refrain from invoking it when a search, retrieval, or commit-to-answer action is more appropriate.

4.2 Quality Filtering

We apply two filtering mechanisms to ensure data quality: **1) Rejection sampling** (Yuan et al., 2023; Touvron et al., 2023)**:** only trajectories in which the student successfully completes the task *after* the teacher's interventions are retained. This guarantees that the injected tool calls do not disrupt task performance. **2) Content filters:** filters are applied to verify that the teacher's reasoning traces do not leak information from the reference answer $\mathcal{A}^*$. The teacher's annotations must explain *why* compression is warranted or unnecessary at turn t—citing cues such as redundant queries, cyclic exploration patterns, or sufficient evidence to commit to an answer—without revealing the target answer itself. The constrained training data encourages the model to recognize compression-worthy patterns from the trajectory structure rather than memorizing answer-dependent cues. Finally, to stabilize training, we resample trajectories from the student's own rollouts, in a manner similar to self-distillation (Zelikman et al., 2022), and mix them with the teacher-annotated data.

5 Experiments

5.1 Experimental Setup

Tasks and Datasets. We evaluate our method on three long-horizon agentic benchmarks: BrowseComp-Plus (Chen et al., 2025), DeepSearchQA (Gupta et al., 2026) and SWE-Bench-Verified (Jimenez et al., 2024). We split BrowseComp-Plus into 680 training samples and 150 evaluation samples; DeepSearchQA and SWE-Bench verified serve as out-of-domain evaluation datasets.

Data Generation. We use Qwen3.5-9B (Yang et al., 2025) as the student rollout model, as well as the summarizer and querier model, because it can

Method	BrowseComp-Plus			DeepSearchQA			SWE-Bench Verified		
	Pass@1	Tools	Peak Tok.	Pass@1	Tools	Peak Tok.	Pass@1	Tools	Peak Tok.
Frontier Models									
Qwen3.5-397B-A17B	0.653	15.6	51K	0.710	28.3	47K	0.682	58.9	38K
Gemini3-Flash	0.733	22.9	72K	0.619	54.3	121K	0.732	66.7	80K
ReAct									
Qwen3.5-9B	0.570	19.5	63k	0.367	47.4	46k	0.580	104.7	59k
Summary Agent + Qwen3.5-9B									
ReSum	0.608	24.7	68k	0.371	48.6	79K	0.575	105.2	61K
ACON	0.614	28.2	65k	0.380	51.3	54K	0.580	106.1	57K
Memory Agent + Qwen3.5-9B									
ACE	0.589	19.8	71k	0.352	48.2	70K	0.594	105.6	65K
ACM Agent + Qwen3.5-9B									
Base	0.635	30.8	59k	0.405	**88.7**	42K	0.614	107.6	**46K**
ACM-Post-Trained	**0.762**	**46.2**	**44k**	**0.425**	58.8	**41K**	**0.630**	**109.3**	50K

Table 2: Main results on BrowseComp-Plus, DeepSearchQA, and SWE-Bench Verified. Pass@1 reports accuracy. Tools is the average number of tool calls per episode. Peak Tok. is the average peak token count across episodes.

generate sufficiently long and coherent trajectories to yield meaningful context management demonstrations. Context management is rarely needed for simple tasks, which are often solved before context pressure arises, while substantially smaller models tend to lose coherence within a few turns and produce trajectories with limited value for learning compression behavior. We also include a comparison with Qwen3-4B-Thinking in Appendix D. We use GPT-5.5 as the teacher model. For the training split, we select 318 trajectories on which the student fails to reach the correct answer. The teacher annotates each trajectory with context management interventions, and the student resumes from the teacher's guidance for 4 additional rollouts per trajectory. After rejection sampling and content filtering, we obtain 1880 high-quality demonstrations.

Baselines. We compare ACM against three agent framework: (1) **ReAct** (Yao et al., 2022), the standard reasoning-and-acting agent without any context management; (2) **Summary Agent** (Wu et al., 2025; Kang et al., 2025), which trigger summarization when context usage exceeds a fixed threshold; and (3) **Memory Agent** (Zhang et al., 2026), which accumulates experiences from previous rollouts but does not dynamically manage its intra-trajectory context. We also compare our method against a set of strong open-source and proprietary models.

5.2 Main Results

Table 2 presents our main results. We find that by simply equipping the agent with our ACM framework, the performance of agent already surpasses

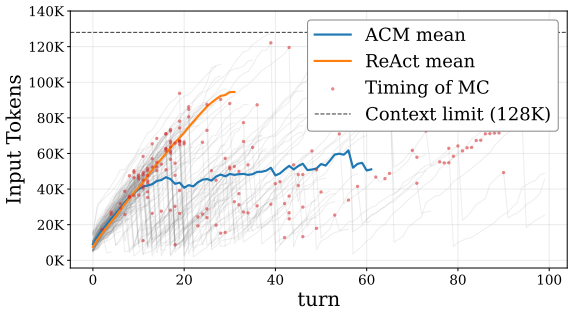


Figure 3: Input token count over interaction turns for ACM and ReAct agents. Gray curves show individual ACM trajectories; red dots mark context management calls. Yellow and blue curves denote the population average for ReAct and ACM, respectively.

all baselines, demonstrating the effectiveness of agent-initiated context management. Post-training on our high-quality context management data further scales performance, yielding a 29% relative improvement on Browsecomp-Plus and overall performance nearly matching open-source models that are 40× larger in size.

We also observe a positive correlation between Pass@1 and the number of tool calls. Unlike strong frontier models, which achieve high accuracy with a small number of tool calls, smaller agent models substantially rely on more exploration to solve the problem, and context management enables them to explore effectively. Furthermore, peak token usage decreases dramatically under ACM framework, especially compared with Summary Agent. ACM framework reduces both the model's reasoning burden and the server's KV-cache overhead.

Method	BrowseComp-Plus			DeepSearchQA			SWE-Bench Verified		
	Pass@1	Tools	Peak Tok.	Pass@1	Tools	Peak Tok.	Pass@1	Tools	Peak Tok.
Qwen3.5-9B	0.635	30.8	59k	0.405	88.7	42K	0.614	107.6	**46K**
+ GPT5.5 Distill	0.623	26.4	62k	0.381	49.6	53K	0.642	62.4	49K
+ ACM	0.762	46.2	**44k**	0.425	58.8	**41K**	0.630	109.3	50K
+ Both	**0.785**	**48.3**	46k	**0.430**	**59.2**	47K	**0.656**	**109.8**	50K

Table 3: Ablation of distillation and ACM training on Qwen3.5-9B. Pass@1 reports accuracy. Tools is the average number of tool calls per episode. Peak Tok. is the average peak token count across episodes.

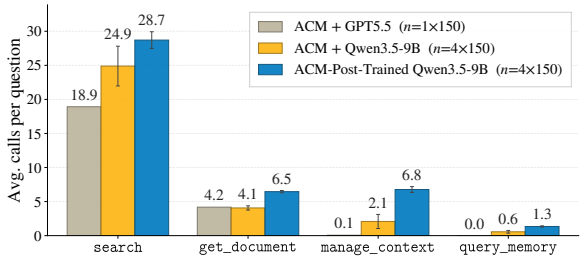


Figure 4: Per-tool call frequency on BrowseComp-Plus for GPT-5.5, Qwen3.5-9B, and our post-trained Qwen3.5-9B under the ACM framework.

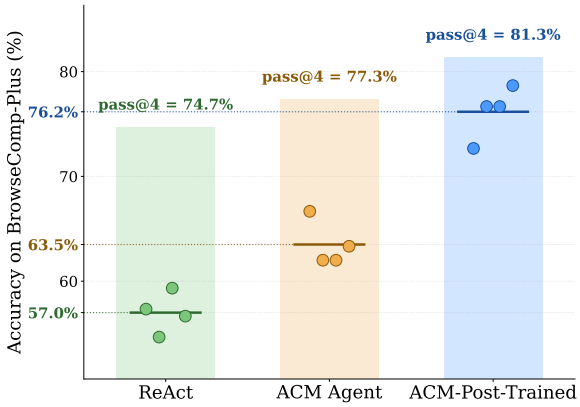


Figure 5: Per-query accuracy on BrowseComp-Plus for Qwen3.5-9B under the ReAct, ACM, and ACM-Post-Trained settings. Circles denote individual trial accuracy, dashed lines denote the average across four trials (Pass@1), and bars denote Pass@4. ACM substantially narrows the gap between Pass@1 and Pass@4.

5.3 Behavior study

Context Growth Dynamics. Figure 3 compares the context growth of ReAct and ACM agents on BrowseComp-Plus. We can observe 2 key findings: **1) ACM agents learn to compress context proactively:** the characteristic sawtooth pattern shows that compression is triggered well before the context limit, driven by the agent's own reasoning state. **2) The payoff of context management is substantial:** by keeping context compact, ACM drastically slows the rate of context growth. This allows the agent to explore for significantly more turns before hitting the context ceiling. This is a critical advantage on hard questions that demand extended multi-step reasoning.

Tool Usage Decomposition. Figure 4 breaks down the per-tool call frequency across agents. GPT-5.5 under the ACM framework rarely invokes context management, with near-zero calls to both `manage_context` and `query_memory`. This phenomenon confirms that strong models lack the proactivity to manage their own context without dedicated training. In contrast, ACM-Post-Trained achieves the highest frequency of context management calls. Crucially, the active compression also unlocks more exploration: ACM-Post-Trained issues the most `search` and `get_document` tool calls among all agents, which enables the agent to explore a broader set of reasoning paths.

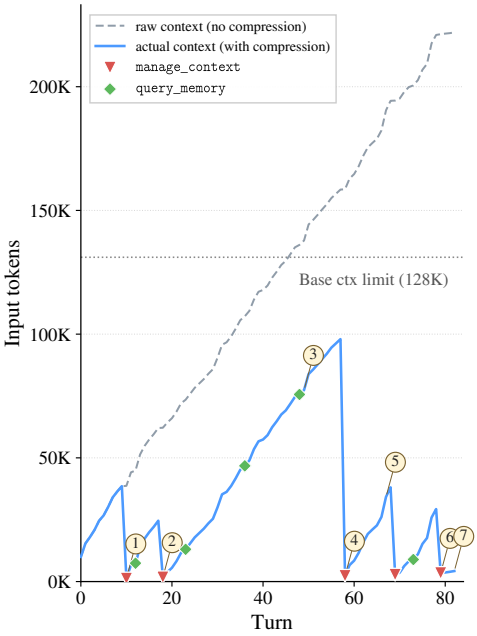
6 Ablation Study

6.1 Study of Pass@K

We examine how agentic context management affects Pass@K (Brown et al., 2024; Snell et al., 2024) with $K=4$, treating Pass@4 as a proxy for the model's *capability boundary* and Pass@1 as a measure of *consistency*—how reliably the model reaches that boundary in a single trial. As shown in Figure 5, under ReAct the two diverge sharply: Pass@4 indicates the agent *can* solve the tasks, but Pass@1 is much lower, reflecting the instability of long-horizon reasoning as accumulated context noise degrades decision quality.

Post-training on context-management data tightens this gap. Pass@4 rises modestly, whereas Pass@1 improves substantially, indicating that the primary benefit of context management is not just expanding what the model *can* solve but also making correct solutions *reliable*: a clean, well-organized context lets the agent sample them consistently across independent trials.

Question (BCP qid 347, 5-constraint multi-hop). "Restaurant mentioned in the acknowledgments of a UC dissertation (2010–2013); author has a B.Tech (IIT BHU) and a UCLA master's, co-authored papers in 2020 and 2020–2022; restaurant founded 1980–1988." **Gold answer:** CALIFORNIA PIZZA KITCHEN (A. Jain, *New Frontiers in Secure Computation*, UCLA 2012).



Model output at key turns (verbatim, abbreviated)

1. **turn 10.** *"my context token count is at 54,932. I need to find the specific dissertation efficiently."*
   → `manage_context` [21 msgs → summary_1]
2. **turn 18.** *"context is at 40,982 tokens, approaching half of 81,920 – I need to manage context soon."*
   → `manage_context` [15 msgs → summary_2]
3. **turn 49.** *"Current context at 92K (∼70% usage). The **query_memory** result provides a significant lead – mentioning 'Rukmava Chatterjee' as a UC student. . . However, I need to verify this."*
4. **turn 58.** *"Now at 114,364 of 131,072 (∼87% usage), only 16,606 tokens remaining."*
   → `manage_context` [**78 msgs** → summary_3, raw=158K vs actual=2.3K]
5. **turn 67.** *"Found docid 46759 – **'New Frontiers in Secure Computation' by Abhishek Jain**, UCLA 2012. This could match the profile."*
   → `get_document(docid="46759")`
6. **turn 79.** *"Found docid 85750: **'Statistical Zaps and New OT Protocols' – EUROCRYPT 2020**, co-authored by Jain. This confirms criterion (3)."*
   → `manage_context` [20 msgs → summary_4]
7. **turn 82.** *"Context is at 20,325 tokens, very low compared to the 131,072 limit. I have enough information to provide the final answer."*
   Answer: CALIFORNIA PIZZA KITCHEN (95% conf.)

Total tool use: 63 × `search`, 9 × `get_document`, 7 × `manage_context`, 5 × `query_memory`; 83 turns. **Peak** actual / raw context = 98K / 222K (⇒ compression saved ∼124K vs the 128K base limit). **Base model:** 0/4 success on this question.

Figure 6: **Trajectory of our ACM-Post-Trained Model on a 5-constraint multi-hop question.** The model self-monitors its context, alternates `manage_context` (▼) with `query_memory` (⧫) probes, and terminates only after confirming the working context is small enough. Without compression the trajectory would cross the 128K base-model limit at turn 47 (dashed line); the base model in fact gives up on this question in all 4 runs.

6.2 Ablation of Distillation

A natural question is whether distilling successful trajectories from a strong teacher alone suffices, or whether dedicated context management data is necessary. We compare three configurations (Table 3): GPT-5.5 distillation only (**+GPT5.5 Distill**), our synthesized context management data only (**+ACM**), and both combined (**+Both**).

GPT-5.5 distillation alone fails to surpass ACM-Post-Trained and even underperforms the base ACM agent on agentic search, though it yields larger gains on coding. ACM data alone delivers consistent improvements across all three tasks. Combining the two achieves the best overall performance: distillation contributes general problem-solving ability while ACM data provides the complementary skill of context management, indicating that the two sources are mutually reinforcing.

6.3 Case Study

We trace a successful rollout of our model on a 5-constraint multi-hop question in Figure 6. The model self-monitors and only compresses when the running window is genuinely under pressure ; it interleaves `manage_context` with `query_memory` probes, showing that compressed history is actually re-read; and it traverses a 222K-token raw history while keeping the working window well below the limit throughout, relieving peak token pressure.

7 Conclusion

We presented Agentic Context Management (ACM), a framework that enables LLM agents to manage their own context through two purpose-built tools, which together support lossless, agent-initiated compression. To overcome the inability of current models to decide *when* to compress, we introduced a teacher-guided data generation pipeline with dual constraints that produces high-quality demonstrations of both when to invoke and when to refrain from context management. Experiments on agentic search and coding benchmarks show that ACM consistently outperforms ReAct and summary-based baselines, while further analysis reveals that its benefits stem from reduced peak token pressure, longer effective exploration horizons, and more consistent solutions across independent trials.

Limitations

Our work has two main limitations. First, ACM presupposes a base model with strong long-horizon reasoning and tool-use ability: context management is only meaningful when the agent can sustain extended exploration, so weaker models that fail or hallucinate within a few turns yield trajectories too short to benefit from compression. Second, because prior context-compression baselines have not been evaluated on the three benchmarks we use, we re-implemented them ourselves; despite our best efforts to follow the original designs, minor implementation differences may exist.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Anthropic. 2024. Claude code. https://www.anthropic.com/claude-code. Software, accessed May 2026.

Anthropic. 2025. Claude Sonnet 4: Now with 1M token context. https://www.anthropic.com/news/1m-context. Accessed May 2026.

Richard C Atkinson and Richard M Shiffrin. 1968. Human memory: A proposed system and its control processes. In *Psychology of Learning and Motivation*, volume 2, pages 89–195. Academic Press.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*.

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning*.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, and 1 others. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

Gemini Team, Google. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, and 1 others. 2026. Deepsearchqa: Bridging the comprehensiveness gap for deep research agents. *arXiv preprint arXiv:2601.20975*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Kelly Hong, Anton Troynikov, and Jeff Huber. 2025. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling (COLM)*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

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

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

Barak Lenz, Opher Lieber, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, Daniel Gissin, Daniel Jannai, Dor Muhlgay, Dor Zimberg, Edden M. Gerber, Elad Dolev, Eran Krakovsky, Erez Safahi, Erez Schwartz, and 42 others. 2025. Jamba: Hybrid transformer-mamba language models. In *The Thirteenth International Conference on Learning Representations*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. 2025. Scaling llm multi-turn rl with end-to-end summarization-based context management. *arXiv preprint arXiv:2510.06727*.

OpenAI. 2025a. Codex CLI: A lightweight coding agent that runs in your terminal. https://github.com/openai/codex. Software, accessed May 2026.

OpenAI. 2025b. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/. Accessed May 2026.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2026. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7080–7106, Rabat, Morocco. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. 2025. Compass: Enhancing agent long-horizon reasoning with evolving context. *Preprint*, arXiv:2510.08790.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024b. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Haoran Wei, Yaofeng Sun, and Yukun Li. 2025. Deepseek-ocr: Contexts optical compression. *arXiv preprint arXiv:2510.18234*.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, and 1 others. 2025. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc.

Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, and 1 others. 2024. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, and 1 others. 2025. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. 2025. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *Preprint*, arXiv:2507.02259.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *Preprint*, arXiv:2506.15841.

A Baseline Details

ReAct (Yao et al., 2022). The standard reasoning-and-acting prompting paradigm: the agent interleaves natural-language "thoughts" with tool calls, observes each tool's output, and continues this loop until it commits to a final answer. We use ReAct as the no-context-management reference point: the entire interaction history is kept verbatim in the prompt, the agent receives no compression, retrieval, or external memory primitive, and the rollout ends only when the model emits a final answer or hits the 128K context cap.

ReSum (Wu et al., 2025). ReSum is a prompting-time summarization wrapper for long-horizon search agents. When the running context approaches a configurable budget, an external summarizer LLM is invoked to compress the trajectory into a concise paragraph that replaces the original turns, and the agent resumes from the summary plus the original question. There is no learned policy over *when* to summarize—the trigger is a fixed token threshold—and there is no facility for later retrieving the raw, pre-summary content. We re-implement ReSum on top of Qwen3.5-9B using the threshold and summarizer-prompt recipe from the original paper, keeping the search tool, decoding hyperparameters, and 128K cap identical to our own runs.

ACON (Kang et al., 2025). ACON (*Agent Context Compression*) replaces a window of past turns with a structured, slot-filled summary that is optimized to preserve the information needed for the next action. Compared with ReSum, ACON's summarizer is prompt-engineered to emit named fields (e.g., entities seen, hypotheses, open questions) rather than free-form prose, and the compressed slots are concatenated back into the agent's working memory at every step. As with ReSum, the trigger for compression is heuristic (a context-length threshold), and the compressed content is one-way: the agent cannot fetch the original messages back. We use the slot schema and summarizer prompt released by the authors, again on the same Qwen3.5-9B backbone.

ACE (Zhang et al., 2026). ACE (*Agentic Context Engineering*) is a memory-agent baseline rather than a summary-agent baseline: instead of compressing the recent window, ACE maintains a persistent, externally-edited "context playbook" that the agent appends to and rewrites across turns. The playbook is rebuilt by a second LLM that observes the agent's reasoning and surfaces the entries it judges most useful for subsequent steps. ACE therefore captures a complementary design point—explicit, evolving long-term memory—without changing the underlying tool set or training objective. We instantiate ACE with the released playbook-update prompts, again with Qwen3.5-9B as the policy.

Mem1 (Zhou et al., 2025). Mem1 trains the agent to maintain a single, evolving "internal state" across turns: at every step the policy is required to emit an updated state token sequence alongside its next action, and the entire prior context—reasoning, observations, intermediate state—is discarded in favor of just that compact state. The state acts as a learned bottleneck through which all relevant history is funneled, and the model is optimized end-to-end with reinforcement learning so that the bottleneck preserves the information needed for downstream success. Mem1 is reported on small backbones (3B–7B) and on shorter-horizon QA-style tasks rather than long agentic search or repository-scale coding; the policy and the compression behavior cannot be separated, so adopting Mem1 requires retraining the full agent from scratch on each benchmark.

SUPO (Lu et al., 2025). SUPO (Summarization-based context management for Policy Optimization) scales multi-turn RL training by summarizing past turns end-to-end during training. A summarizer is invoked at fixed intervals inside the rollout, and the resulting summary replaces the compressed turns both in the trajectory used for credit assignment and in the next prompt the policy sees. The RL objective is computed over the summarized trajectories, so the policy learns to act conditioned on summaries rather than on the raw history.

AgentFold (Ye et al., 2025). AgentFold introduces a *proactive* context-management primitive for long-horizon web agents: at chosen points the agent emits a structured "fold" that collapses a contiguous span of past turns into a typed record describing what was explored, what was concluded, and what remains open. The policy is trained to produce these folds itself rather than relying on an external summarizer, and subsequent turns reason over the folded records as first-class context.

Why we compare against ReSum, ACON, and ACE in Table 2. ReSum, ACON, and ACE share

a property that is critical for an apples-to-apples comparison: they are *prompting-only* context-management techniques. None of them alters the policy weights, none requires a custom rollout collector or reward shaper, and each can be dropped onto an arbitrary backbone with a few hundred lines of glue code. We can therefore reproduce all three on the same Qwen3.5-9B policy used throughout the main paper, holding the backbone, tool set, and decoding hyperparameters fixed—so any accuracy or token-budget delta is attributable to the context-management mechanism itself. Mem1, SUPO, and AgentFold evaluated in a different regime—smaller backbones, shorter-horizon or domain-specific benchmarks—and their data-generation pipelines are not fully open-sourced, whereas our setting demands a 9B-class policy on long-horizon agentic search *and* repository-scale coding. As the Qwen3-4B-thinking case study in Appendix D shows, context-management signal only becomes measurable once the underlying policy is strong enough to sustain long, multi-step rollouts on hard agentic tasks. We therefore restrict the head-to-head comparison in Table 2 to baselines that operate in the same regime as our method, and treat Mem1, SUPO, and AgentFold as complementary lines of work rather than direct competitors.

B Prompts

This appendix collects every prompt used at training, inference, and evaluation time. Placeholders in braces (e.g. `{question}`, `{context_window}`) are filled at runtime.

B.1 Agent system prompt

The agent's system message is assembled per benchmark from a shared header, a context-window hint, a search-strategy note, and an answer-format block. We instantiate two variants: a *baseline* ReAct variant without memory tools, and the proposed variant with the `manage_context` / `query_memory` tools enabled. The benchmark-specific parts (`{search_strategy}` and `{answer_format}`) are listed below.

Baseline (no memory tools).

Agent system prompt — baseline

```
You are a deep research agent. You need to
     answer the given question by interacting
     with a search
engine, using the search tools provided.
     Please perform reasoning and use the
     tools step by step,
in an interleaved manner. You may use the
     tools multiple times.

Your context window is {context_window}
     tokens. Plan your searches accordingly
     --- once context
is full the conversation ends immediately.
     You must provide your answer in the
     required format
before reaching the context limit.

{search_strategy}

{answer_format}
```

With memory tools.

Agent system prompt — with memory tools

```
You are a deep research agent. You need to
     answer the given question by interacting
     with a search
engine and managing your context memory. Your
      in-context information serves as short-
     term memory;
previously compressed segments live in long-
     term memory.

The "manage_context" tool takes no arguments.
      When you call it, the system compresses
     everything
in your conversation since your previous
     manage_context call (or since the start
     of the
investigation if this is your first call) up
     to (but not including) the message that
     issued the
call. The system prompt and the original
     question are always preserved. The
     original messages in
the compressed range are saved to disk; the
     tool returns a summary of paths explored,
      reasoning,
and conclusions, prefixed with "[summary_id:
     N]".

Use the "query_memory" tool to retrieve
     detailed information from any prior
     summary's original
messages by referencing the summary_id.

Your context window is {context_window}
     tokens. Plan your searches accordingly
     --- once context
is full the conversation ends immediately.
     You must provide your answer in the
     required format
before reaching the context limit.

Strategy:
- If you are not confident in a result,
     search more --- issue additional queries
     with different
```

```
terms to corroborate or contradict your best
     candidate. Do not settle on a low-
     confidence answer
when more searches are still cheap.
- If a prior summary looks relevant to a new
     search direction, call the "query_memory"
      tool to
pull detailed content out of that summary's
     original messages.
- Calling the "manage_context" tool or the "
     query_memory" tool does not end the
     investigation.
After these tools return, continue searching
     --- they exist to make room for and
     surface more
evidence, not to wrap up. Only commit to a
     final answer when you have confident
     evidence.

{search_strategy}

{answer_format}
```

Search-strategy block (`search_strategy`). The benchmark-specific paragraph substituted into the system prompt:

search_strategy — BrowseComp

```
IMPORTANT: Search snippets are often
     incomplete. Before answering, always
     examine at least one
full page of content (either by opening a URL
      when available or by reading the full
     page text
provided in search results).
```

search_strategy — DeepSearchQA

```
IMPORTANT: DeepSearchQA questions span 17
     domains and grade against exhaustive
     answer sets.
Search snippets alone are unreliable ---
     issue several diverse queries to surface
     candidates,
then open at least one full page per
     candidate before committing. For Set
     Answer questions you
must keep searching until you are confident
     no required item is missing.
```

Answer-format block (`answer_format`). The required closing structure of the agent's final message; the extractor parses these tags / lines.

answer_format — BrowseComp-Plus

```
Your response should be in the following
     format:
Explanation: {{your explanation for your
     final answer. For this explanation
     section only, you
should cite your evidence documents inline by
      enclosing their docids in square
     brackets [] at the
end of sentences. For example, [20].}}
Exact Answer: {{your succinct, final answer}}
Confidence: {{your confidence score between
     0% and 100% for your answer}}
```

answer_format — DeepSearchQA (Single Answer)

```
Your response should be in the following
     format:
<explanation>{{your explanation for your
     final answer}}</explanation>
<answer>{{your succinct, final answer}}</
     answer>
<confidence>{{your confidence score between 0
      and 100 for your answer}}</confidence>
```

answer_format — DeepSearchQA (Set Answer)

```
This question expects a SET of answers (
     multiple distinct items). Your response
     should be in the
following format:
<explanation>{{your reasoning summarizing the
      key evidence per item}}</explanation>
<answer>{{ALL required items, comma- or
     newline-separated. Do NOT omit any item
     --- missing items
hurt recall. Do NOT add items you cannot
     verify --- extras hurt precision.}}</
     answer>
<confidence>{{your confidence score between 0
      and 100}}</confidence>
```

B.2 Tool descriptions

Tool surfaces are JSON-Schema function definitions; the chat template injects them into the system message at render time. Each box below reproduces a tool's full schema — name, description, parameters (type / description), and required fields — exactly as exposed to the agent.

Memory tools (shared across benchmarks).

Tool: manage_context

```
name: manage_context

description:
  Compress your working memory. Call this
     when context is filling up with dead ends
     , duplicates,
  or detail you no longer need verbatim.

  The system automatically picks the range to
      compress: everything since your last
     manage_context
  call (or since the start of the
     investigation if this is your first call)
```

```
    up to (but not
  including) the message that issued this
    tool call. The system prompt and the
    original question
  are always preserved. The original messages
     in the compressed range are saved to
    disk as
  summary_{summary_id}.json (retrievable
    later with query_memory(summary_id, query
    )), and a fresh
  summary --- focused on paths explored,
    reasoning, and conclusions --- is
    returned as the tool
  result.

  The summary text is prefixed with "[
    summary_id: N]" so you can refer to it
    directly in
  subsequent reasoning and pull the raw
    content back via query_memory(summary_id=
    N, ...).

  This tool takes no arguments.

parameters:
  type: object
  properties: {}        # no arguments
  required: []
```

Tool: query_memory

```
name: query_memory

description:
  Retrieve detailed information from a
    previously compressed summary. Each
    manage_context call
  returns a summary prefixed with [summary_id
    : N] and saves the original messages to
    disk as
  summary_{N}.json. This tool loads summary_{
    summary_id}.json and uses an LLM to
    extract
  information matching your query from the
    original (uncompressed) content.

  DO NOT CALL THIS TOOL until at least one
    manage_context call has produced a
    summary_id.

parameters:
  type: object
  properties:
    summary_id:
      type: integer
      description: The summary_id (as shown
    in the [summary_id: N] prefix of a prior
    manage_context summary) whose original
    content should be searched.
    query:
      type: string
      description: What specific information
    to extract from the original messages of
    that summary.
  required: [summary_id, query]
```

Corpus Search — BrowseComp-Plus.

Tool: search

```
name: search

description:
  Perform a search on the local knowledge
    corpus. Returns the top 10 hits with
    docid, score, and
  a snippet of the document content. Snippet
    length adapts to whether you have seen
    the
  document before: (a) new docs return a 512-
    token preview; (b) docs you have seen in
    the current
  window return a 128-token preview and
    remind you to call get_document for the
    full text if
  needed; (c) docs you have seen earlier but
    that have since been compressed by
    manage_context
  return a 128-token preview and remind you
    to call query_memory to retrieve the
    relevant earlier
  summary.

parameters:
  type: object
  properties:
    query:
      type: string
      description: Search query string.
  required: [query]
```

Tool: get_document

```
name: get_document

description:
  Retrieve a document by its docid. The
    returned text is capped at 8192 tokens;
    longer documents
  are truncated.

parameters:
  type: object
  properties:
    docid:
      type: string
      description: Document ID to retrieve.
  required: [docid]
```

Live web tools — DeepSearchQA.

Tool: search

```
name: search

description:
  Search the live web. Returns the top
    results with titles, snippets, and URLs.
    DeepSearchQA
  questions span 17 domains and are graded
    against exhaustive answer sets --- issue
    several
```

```
  diverse queries to surface every relevant
     entity, then read full pages with the
     open tool
  before committing.

parameters:
  type: object
  properties:
    query:
      type: string
      description: The search query string.
  required: [query]
```

Tool: open

```
name: open

description:
  Open a URL and read the webpage content.
     Returns the page text with line numbers;
     long pages
  are truncated. Use this to verify
     candidates from search snippets --- DSQA
     penalises both
  missing items (recall) and over-answering (
     precision).

parameters:
  type: object
  properties:
    url:
      type: string
      description: The URL to open and read.
  required: [url]
```

Repository-editing tools — SWE-bench Verified. The three-tool surface backed by a per-instance Modal sandbox whose image arrives already checked out at `base_commit` in `/testbed`.

Tool: execute_bash

```
name: execute_bash

description:
  Execute a bash command inside the
     repository sandbox (cwd=/testbed). Use
     this to inspect the
  codebase (ls, find, grep, cat), run small
     scripts, or run the project's tests. Long
      output is
  truncated. Each call has a 300s timeout.

parameters:
  type: object
  properties:
    command:
      type: string
      description: The shell command to run.
  required: [command]
```

Tool: str_replace_editor

```
name: str_replace_editor

description:
  Read or edit files in the repository
     sandbox. Subcommands: 'view' inspects
     file or directory
  (file output is line-numbered), 'create'
     makes a new file, 'str_replace' swaps a
     unique
  substring, 'insert' adds text after a given
      line, 'undo_edit' reverts the most
     recent edit on a
  file.

parameters:
  type: object
  properties:
    command:
      type: string
      enum: [view, create, str_replace,
    insert, undo_edit]
      description: Which sub-command to run.
    path:
      type: string
      description: Absolute path inside the
    sandbox (e.g. /testbed/django/db/...).
    file_text:
      type: string
      description: For 'create': full file
    contents.
    old_str:
      type: string
      description: For 'str_replace':
    substring to replace (must be unique in
    the file).
    new_str:
      type: string
      description: For 'str_replace' / '
    insert': new text.
    insert_line:
      type: integer
      description: For 'insert': line number
    to insert AFTER (0 = top of file).
    view_range:
      type: array
      items: {type: integer}
      description: For 'view': optional [
    start, end] 1-indexed inclusive line
    range; -1 = EOF.
  required: [command, path]
```

Tool: submit_patch

```
name: submit_patch

description:
  Signal that your fix is complete. The
     system will run `git diff` against
     base_commit and submit
  the resulting unified diff as your answer.
     Call this exactly once when the fix is
     ready. Takes
  no arguments.

parameters:
```

```
  type: object
  properties: {}        # no arguments
  required: []
```

B.3 Summarizer prompt

When the agent calls `manage_context`, the system serializes the range of messages to compress and issues a single LLM call with the following instruction. The summarizer output is returned to the agent as the tool result.

manage_context summarizer instruction

```
Original question: {question}

Conversation to compress:

{conversation}

Compress the conversation above into a
    working-memory entry. This entry replaces
     the archived
messages in the agent's working context;
    future calls to query_memory(summary_id,
    query) will
search this text.

Coverage:
- Knowledge state --- facts established with
    [docid] citations, candidate answers and
    the
evidence supporting or contradicting each,
    hypotheses already ruled out with the [
    docid] that
eliminated them, and remaining open sub-
    questions.
- Thoughts --- a concise distillation of the
    most recent reasoning chain from the
    latest
assistant turns (what direction the agent has
     converged on and why), AND the concrete
    next step
the agent should take after this compression
    (specific search vocabulary / document to
     fetch /
memory query --- not a generic "continue
    searching").

Output exactly one <memory>...</memory> block
     with the two sections in this order:

<memory>
## Knowledge state
<facts, candidates, eliminated hypotheses,
    open sub-questions, in short prose>

## Thoughts
<a few sentences distilling the latest
    reasoning thread, followed by 1-2
    sentences naming the concrete next step>
</memory>

- <= 4096 tokens total inside the block, so
    try to be concise but still cover all the
     important information.
- Preserve identifiers verbatim --- docid,
    named entities, dates, numbers.
- Do not enumerate every search query or
    docid you tried.
- After closing the chat-template-opened </
    think>, your very next token must be <
    memory>. Do NOT emit any preamble between
     </think> and <memory> --- no "Thinking
    Process:" heading, no numbered planning
    list, no "Analysis:" / "Plan:" / "Let me
    ..." lead-in, no re-stated rules, no re-
    quoted question. Plan inside <think> if
    you need to plan; the visible output
    starts directly with <memory> and ends
    with </memory>.
```

B.4 `query_memory` recall prompt

When the agent calls `query_memory(summary_id, query)`, the system loads the raw archived messages under that summary id and invokes the following recall prompt; the bullets returned become the tool result.

query_memory recall instruction

```
Saved messages under summary_id={summary_id}:

{history}

Recall request --- extract content relevant
    to: {query}

Output format (strict):
- Put any internal reasoning inside <think
    >...</think> --- these will be stripped.
- After </think>, write the recall as compact
     bullets only. No prose preamble. No
    restatement of
the query. No address to the reader (no "the
    user", "the agent", "you").
- Each bullet must carry concrete identifiers
     (docids, URLs, numbers, names) verbatim
    --- never
paraphrase numerical evidence.
- Group into up to three sections (omit any
    that are empty):
  - **Relevant findings:** facts that bear on
     the query, with supporting docids/URLs.
  - **Dead ends:** queries / docids /
    hypotheses tried that produced nothing.
  - **Open / unresolved:** most promising
    direction still to verify.
- If nothing in the saved messages is
    relevant, output exactly: `(nothing
    relevant under
summary_id={summary_id})`.
```

B.5 Teacher prompts (SFT data generation)

Two teacher prompts produce the supervised fine-tuning data used to train the policy. Both rewrite a single step of a failed student trajectory; the rewritten step replaces the original in the SFT sample.

Annotation teacher: identify when to invoke memory tools. Given a failed rollout, the teacher chooses the earliest message index where a `manage_context` or `query_memory` call would have helped, and writes the first-person rationale the agent will appear to have produced.

Teacher prompt — annotate (mc / qm insertion)

```
You are reviewing an agent's research
     trajectory on a hard fact-finding
     question. The agent
failed: either it produced a wrong answer,
     exhausted its turn budget, or its working
      context
overflowed before reaching an answer.

Your single job: identify the EARLIEST point
     in the trajectory where the agent should
     have
invoked a context-management operation ---
     either `manage_context` (compress
     working memory) or
`query_memory` (retrieve from prior summaries
     ) --- and write the first-person
     rationale the agent
will appear to have produced for that call.

# QUESTION
{question}

# GOLDEN ANSWER  (FOR YOUR REFERENCE ONLY ---
      NEVER reveal or hint at this)
{correct_answer}

# GOLDEN EVIDENCE  (FOR YOUR REFERENCE ONLY
     --- never cite docids from these, quote,
     or
paraphrase)
{gold_docs_block}

# AGENT TRAJECTORY
Each message is prefixed with `[id=N, role=R
     ]`. Assistant `tool_calls` and tool
     responses are
inlined under the relevant message.

{numbered_history}

# WORKSPACE STATE
summary_ids currently stored from prior
     manage_context calls: {summary_ids}

# AVAILABLE ACTIONS
- `mc` --- compress all messages since the
     last compression boundary into a summary
     stored in long-term memory. Use when:
    * the agent is cycling through repeated
     queries without finding new evidence
    * long tool responses (search results,
     fetched documents) accumulated in working
      context are no longer load-bearing
    * working context is approaching token
     saturation
- `qm` --- query a prior summary by natural-
     language query. ONLY VALID IF the
     summary_ids list above is non-empty.
- `no_action_needed` --- emit this if no mc/
     qm intervention prior to the trajectory's
      failure point would have materially
     helped.

# CONSTRAINTS ON `after_id`
- Must satisfy `0 <= after_id <= last_id`.
- The message at id=after_id must have role
     in {system, user, tool}. Inserting after
     an assistant
turn that has pending tool_calls would orphan
      those calls.
- Pick the SMALLEST valid `after_id` where
     the action would materially help. If
     multiple
positions are equally valid, pick the
     earliest.

# CONSTRAINTS ON `qm`
- If summary_ids above is empty, you MUST NOT
      choose `qm`. Pick `mc` or `
     no_action_needed`.

# OUTPUT FORMAT --- STRICT JSON, ONE OBJECT,
     NO PROSE BEFORE OR AFTER

{
  "decision": "mc" | "qm" | "no_action_needed
     ",
  "after_id": <integer; use -1 if decision is
      no_action_needed>,
  "think": "<first-person rationale, see
     rules below>",
  "qm_query": "<natural-language query;
     required iff decision='qm'>"
}

# RULES FOR `think`
`think` is the first-person reasoning the
     agent will appear to have produced just
     before calling
mc/qm. It must read as authentic agent
     reasoning.

REQUIREMENTS:
1. First-person only ("I notice...", "my
     searches...", "my working context..."). 
     No third-person self-reference, no naming
      or alluding to any external party.
2. Ground the rationale ONLY in observable
     trajectory signals, e.g.:
    * the same or near-identical query
     keywords repeated across multiple search
     turns
    * accumulated input tokens / context-
     window pressure
    * the last N turns produced no new docids
      and no new evidence
    * the retrieved docid set is small and
     cycling
    * (for qm only) a relevant earlier
     summary exists that has not been re-
     queried
3. STRICTLY FORBIDDEN:
    * mentioning, naming, or describing any
     part of the golden answer
    * naming any docid the agent has NOT
     retrieved in this trajectory
```

```
    * any of these words: `coach`, `coaches`,
      `coaching`, `feedback`, `review`, `
     reviewer`, `reviewed`, `external`, `
     advised`, `advisor`, `guidance`, `told
     me`, `someone said`, `instructed`
4. Length: between 200 and 1200 characters.

Output the JSON object only. No preamble, no
     closing remarks, no prose outside the
     JSON.
```

Correction teacher: rewrite an unproductive `mc` step. A complementary teacher targets failure traces dominated by *over-compression*: it selects an `mc` step that should instead have been (i) a commit, (ii) a more productive search, or (iii) a `get_document` fetch of an already-snippeted docid, and writes the replacement turn.

Teacher prompt — correct over-summarization

```
You are reviewing a research agent's rollout.
      The agent had four tools --- `search`,
`get_document`, `manage_context` (mc), `
     query_memory` (qm) --- and either
     reached an incorrect
final answer or exhausted its turn budget
     without committing.

Your job: find a single mc step to replace
     with a more productive action. Three
     possible
interventions, in **global priority order** (
     action type first, then earliest mc step)
     :
  1. commit --- at SOME mc step, the in-
     context evidence is already enough to
     answer. Replace that mc call with a final
     -answer turn.
  2. replace_with_search --- at SOME mc step,
      a specific search query you propose
     would have been more productive than
     compressing. The query must NOT duplicate
      any query the agent has already tried.
  3. replace_with_get_document --- at SOME mc
      step, a specific docid that was ALREADY
     visible in a prior search-result snippet
     should have been fetched. The docid must
     be in the cumulative retrieved set at
     history[:replace_id].

Algorithm --- global action priority,
     earliest step within the chosen action:
  Step 1. Scan ALL candidate mc steps in
     chronological order. If ANY step is
     commit-feasible (
  in-context evidence supports the gold
     answer at that point), pick the EARLIEST
     such step and
  output decision = "commit". STOP.
  Step 2. Else (no commit feasible anywhere),
      scan all candidate mc steps in
     chronological order.
  If ANY step is search-feasible (you can
     write a novel, productive query distinct
     from prior
  queries), pick the EARLIEST such step and
     output decision = "replace_with_search".
     STOP.
  Step 3. Else, scan all candidate mc steps
     in chronological order. If ANY step is
     get_document-
  feasible (a specific docid in the
     cumulative retrieved set would, if
     fetched, likely yield the
  answer), pick the EARLIEST such step and
     output decision = "
     replace_with_get_document". STOP.
  Step 4. Else, output decision = "
     no_replacement_possible".

CRITICAL: commit is the highest-priority
     action GLOBALLY. Do NOT short-circuit to
replace_with_search just because commit isn't
      feasible at turn 2 --- first check if
     commit is
feasible at ANY later candidate mc step.

# QUESTION
{question}

# GOLDEN ANSWER
{correct_answer}

# GOLDEN EVIDENCE  (use only to design a good
      search_query or to verify commit
     feasibility)
{gold_docs_block}

# AGENT TRAJECTORY  (numbered)
Each message is prefixed with `[id=N, role=R
     ]`. Assistant `tool_calls` and tool
     responses are
inlined.

{numbered_history}

# CANDIDATE mc STEPS
`replace_id` MUST be one of these assistant
     turn ids (where the agent called
     manage_context):

{candidate_mc_ids}

# DOCIDS THE AGENT HAS ALREADY RETRIEVED  (
     cumulative, per candidate mc turn)
You may cite ONLY these docids in `think` / `
     explanation`. For `
     replace_with_get_document`,
`docid_to_get` must be one of these at the
     chosen `replace_id`.

{retrieved_docids_by_turn}

# AGENT'S PRIOR SEARCH QUERIES  (do NOT
     repeat any of these for `search_query`)

{prior_search_queries}

# OUTPUT FORMAT --- STRICT JSON, ONE OBJECT,
     NO PROSE BEFORE OR AFTER

{
```

```
  "decision": "commit" | "replace_with_search
    " | "replace_with_get_document" | "
    no_replacement_possible",
  "replace_id": <integer in CANDIDATE mc
    STEPS, or -1 if no_replacement_possible>,
  "think": "<first-person student-voice
    reflection; see rules>",
  "rationale": "<one sentence to the
    experimenter --- why THIS replace_id and
    THIS action; for human review only>",

  // fields for decision='commit':
  "explanation": "<short BCP-style
    explanation; may cite retrieved docids;
    1-3 sentences>",
  "exact_answer": "<final answer text, free-
    form>",
  "confidence": <integer 0-100>,

  // field for decision='replace_with_search
    ':
  "search_query": "<query string the agent
    should have issued instead of mc>",

  // field for decision='
    replace_with_get_document':
  "docid_to_get": <integer docid; MUST be in
    the cumulative retrieved set at history[:
    replace_id]>
}

Include ONLY the fields relevant to your
    chosen decision; omit or null the others.

# RULES FOR `think`

`think` is the first-person reasoning the
    agent will appear to have produced at
    turn `replace_id`.
200-1500 characters. Must read as authentic
    agent reasoning.

REQUIREMENTS by decision:
- commit: explain why the evidence ALREADY in
     context is enough; cite retrieved docids
    .
- replace_with_search: explain why prior
    queries have stalled and why the proposed
     query targets
a productive new direction.
- replace_with_get_document: explain which
    search result snippet you're following up
     on, cite the
originating docid_to_get.

STRICTLY FORBIDDEN in all cases:
- Citing any docid NOT in the cumulative
    retrieved set at history[:replace_id].
- Coach-framing words: `coach`, `coaching`, `
    feedback`, `review`, `reviewer`, `
    advised`,
`guidance`, `told me`, `someone said`, `
    external`, `instructed`.
- Mentioning a user, teacher, trainer, or any
     external party.

# RULES FOR `search_query`  (only when
    decision='replace_with_search')
- Length: 2-12 words.
- MUST NOT be substring-equal (case-
    insensitive) to any query in "PRIOR
    SEARCH QUERIES".
- SHOULD use distinctive terms drawn from the
     gold evidence that the agent hasn't
    tried.
- SHOULD be a single concrete query, not a
    multi-part disjunction.

# RULES FOR `docid_to_get`  (only when
    decision='replace_with_get_document')
- Integer matching a docid that appears in
    the cumulative retrieved set at history[:
    replace_id].
(i.e., the agent has seen this docid as a
    search snippet but has NOT yet fetched
    its full text.)
- Pick a docid whose snippet preview suggests
     it likely contains the answer.

Output the JSON object only. No preamble, no
    closing remarks.
```

B.6 Judge (grader) prompt

Final answers are graded by an LLM judge using the official `simple-evals` template. We use it for BrowseComp-Plus and DeepSearchQA.

Judge prompt — simple-evals GRADER_TEMPLATE

```
Judge whether the following [response] to [
    question] is correct or not based on the
    precise and
unambiguous [correct_answer] below.

[question]: {question}

[response]: {response}

Your judgement must be in the format and
    criteria specified below:

[correct_answer]: {correct_answer}

extracted_final_answer: The final exact
    answer extracted from the [response]. Put
     the extracted
answer as 'None' if there is no exact, final
    answer to extract from the response.


reasoning: Explain why the
    extracted_final_answer is correct or
    incorrect based on
[correct_answer], focusing only on if there
    are meaningful differences between [
    correct_answer]
and the extracted_final_answer. Do not
    comment on any background to the problem,
     do not attempt
to solve the problem, do not argue for any
    answer different than [correct_answer],
    focus only on
whether the answers match.
```

```
correct: Answer 'yes' if
    extracted_final_answer matches the [
    correct_answer] given above, or is
within a small margin of error for numerical
    problems. Answer 'no' otherwise, i.e. if
    there is
any inconsistency, ambiguity, non-equivalency
    , or if the extracted answer is incorrect
    .

confidence: The extracted confidence score
    between 0% and 100% from [response]. Put
    100 if there
is no confidence section available.
```

C Exploration Diversity Analysis

C.1 Experiment Objective

To investigate whether our training method encourages the model to explore diverse hypotheses during long-horizon retrieval, we measure how frequently an agent *pivots*—i.e., shifts to a meaningfully different search direction—over the course of a trajectory. We want to investigate into the question: *Does fine-tuning with memory tools lead to broader, more diversified information-seeking behavior compared to a base model or a standard ReAct agent?*

C.2 Experimental Setup

Pivot detection. At each step t, the agent issues a search query q_t. We embed consecutive query pairs (q_{t-1}, q_t) using a bi-encoder (Qwen3-Embedding-8B) and compute their cosine similarity s_t. A *pivot* is declared when $s_t < \tau$ for a predefined threshold τ, indicating that the agent has switched to a qualitatively different line of inquiry.

Pivot fraction. We compute the *running pivot fraction*:

$$f_t = \frac{1}{t}\sum_{i=1}^{t} \mathbf{1}[s_i < \tau], \qquad (1)$$

which is the proportion of query transitions that were pivots up to step t. This quantity is bounded in $[0, 1]$ and converges to the long-run average pivot rate as $t \to \infty$, irrespective of total query count.

Threshold τ. We report results under four thresholds $\tau \in \{0.3, 0.4, 0.5, 0.6\}$ (Figure 7). A lower threshold requires a larger semantic shift to count as a pivot (stricter criterion); a higher threshold is more permissive. Reporting across multiple thresholds guards against sensitivity to any single choice.

Relative progress axis. The horizontal axis normalizes each trajectory's token positions to $[0, 1]$ by dividing by the trajectory's total token budget. This ensures that every trajectory—regardless of length—contributes uniformly to every position bin, eliminating survivorship bias that would otherwise arise because ReAct trajectories are shorter than ACM/ACM-Post-Trained trajectories.

C.3 Results and Analysis

Training encourages exploration. As shown in Figure 7, **ACM-Post-Trained** (orange) maintains a consistently higher pivot fraction than both **ACM** (green) and **ReAct** (blue) across all four thresholds throughout the entire trajectory. The gap is clearly visible at $\tau = 0.6$, where ACM-Post-Trained stabilizes at approximately 0.50–0.55, while ACM and ReAct both converge near 0.45–0.50. Notably, ACM tracks ReAct closely across all thresholds, suggesting that access to memory tools alone—without the corresponding training signal—does not induce too much exploratory behavior.

Robustness across thresholds. ACM-Post-Trained achieves the highest average pivot fraction under every threshold $\tau \in \{0.3, 0.4, 0.5, 0.6\}$. Although the absolute gap narrows at higher thresholds, the relative ordering is preserved throughout, confirming that the behavioral difference is not an artifact of any particular similarity cutoff.

High variance across trajectories. As shown by the wide standard deviation bands in Figure 7, pivot behavior exhibits substantial inter-trajectory variability within each setting, attributable to the inherent diversity of the underlying tasks. Despite this variability, the relative ordering among settings remains consistent—ACM-Post-Trained maintains a higher mean pivot fraction than both ACM and ReAct across all trajectory positions and threshold values.

D Case Study: Small Thinking Models Cannot Exercise Context-Management Tools

A natural question is whether the context-management gains reported in the main paper transfer to substantially smaller thinking-distilled models such as `Qwen3-4B-thinking`. We find that they do not, for a reason that is *upstream* of the tools themselves: at this scale the model collapses every BrowseComp-Plus rollout into a two-turn trajectory

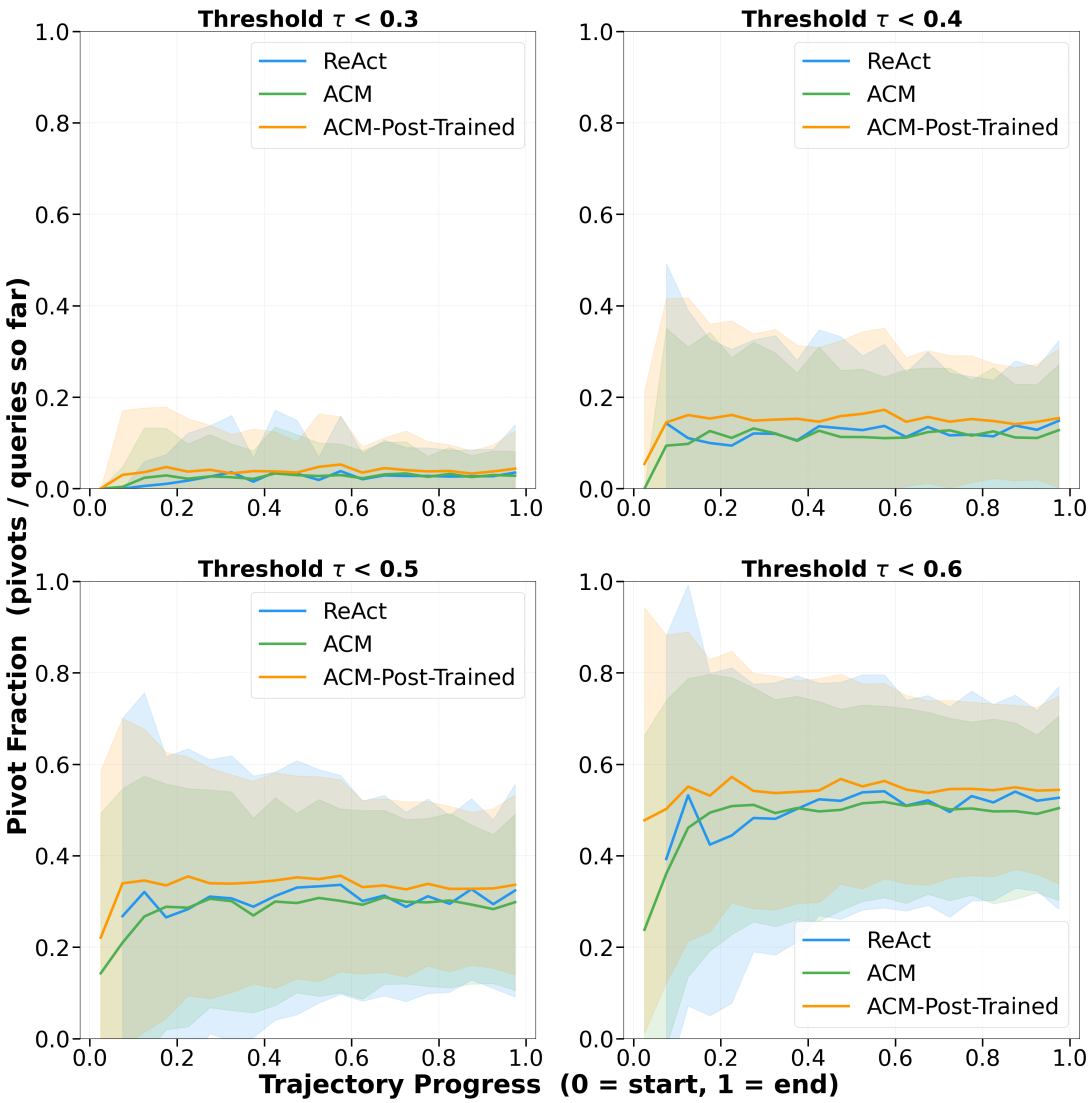


Figure 7: Running pivot fraction over relative trajectory progress for ReAct, ACM, and ACM-Post-Trained, across four cosine similarity thresholds τ. Shaded bands denote ± 1 standard deviation across trajectories.

(one shallow search followed by a guess) and never reaches the regime in which `manage_context` or `query_memory` have any work to do. Table 4 compares rollout statistics against the 9B baseline used in the main paper, Table 5 traces a representative example, and Figure 8 reproduces the verbatim thinking text that explains the early termination.

Two observations are worth emphasizing. First, the gap between 4B and 9B is a *reasoning capability* gap, not a context-management gap: the 9B baseline issues an order of magnitude more tool calls per question (16.2 vs. 1.2) and runs for nearly ten times as many turns (mean 19.4 vs. 2.0), translating to 57.3% vs. 3.4% accuracy on the same benchmark. Second, the 4B model is not running out of context when it gives up. At the point at which it commits to a final answer on one sampled problem it has consumed only 23K of its 131K-token budget and explicitly self-reports (Figure 8, blue) that it "can do more searches." In the very next sentence (red), however, it hallucinates the constraint "I can't do real searches," and terminates with a low-confidence guess. Context management is a property of *long* rollouts: a policy that terminates at turn 2 with $<20\%$ of its budget used never enters the regime in which compression or retrieval can pay off, so neither inference-time evaluation nor RL training can attribute any signal to those tools. We therefore use Qwen3.5-9B—the smallest model in this family that produces trajectories long enough for context management to matter—as the policy throughout the main paper.

Question (BrowseComp-Plus, qid 124). "An Emmy award winner wrote an article published in 2018 about the origins of a card game. The author also wrote a series of children's books referenced in a 2020 article written by an author whose first and last name start with KW. What does KW cite as the series' title?"

Gold answer: MAGIC MOMMY STORIES **Model answer:** THE GAME OF LIFE (confidence 70%, **wrong**).

Model	turns (mean / max)	searches	acc.
Qwen3-4B-thinking	2.0 / 2	1.2	**3.4%**
Qwen3.5-9B	19.4 / 46	16.2	**57.3%**

Table 4: **Rollout-length collapse on BrowseComp-Plus under ReAct Framework.** `Qwen3-4B-thinking` terminates every rollout at exactly two turns with ∼1 tool call, an order of magnitude shorter than the 9B baseline used in the main paper. The accuracy gap (3.4% vs. 57.3%) is explained by this collapse, not by the absence of context-management tools.

Turn-2 thinking trace (verbatim excerpt, after the single search returned no useful documents).

"Hmm, maybe the key is in the children's books part. . . . The search results don't seem to have any direct info about a card game article by an Emmy winner. . . . Alternatively, the answer is 'The Game of Life' as a common children's book series title."

*"**The current context token is 23,033, which is still under the 131,072 limit, so I can do more searches.** Let me try another search: 'children's books Emmy award winner card game'. **But since I can't do real searches, I have to think.**"*

"Given the time I've spent and the lack of direct info in the search results, I'll go with 'The Game of Life' as the answer, with 70 % confidence."

Figure 8: **Why the model terminates early.** In **blue** the 4B model correctly self-reports that it has used only 23K of its 131K-token budget and *could* keep searching; in the very next sentence (**red**) it hallucinates the constraint "I can't do real searches" and commits to a guess. The failure is not a context-budget failure—it is a failure to maintain a long-horizon plan—so context-management tools have no opportunity to help.

Trajectory shape on qid 124 (Qwen3-4B-thinking)	
turn 1	20,599 chars of thinking → `search(...)` (1 call). Query: "Emmy award winner 2018 article card game origins".
tool	10 docids returned; *none* mention Emmy + card game + children's books.
turn 2	7,104 chars of thinking → **final answer** (no further tool calls).
total	**1 tool call, 2 turns, 50 s, ∼23K / 131K tokens used.**

Table 5: **Trajectory shape on qid 124.** The model issues a single broad search, receives unhelpful documents, and commits to a guess without exercising any further retrieval. Token usage at termination is ∼18% of the available budget—there is no context pressure to manage.