R_FOLD: Bi-Level Context Management for Long-Horizon Tool-Using Agents

Anonymous ACL submission

Abstract

Long-horizon tool-using agents need to retain task-relevant evidence from verbose observations while operating under finite context budgets. This remains challenging even for long-context models, since practical serving configurations may expose substantially smaller windows than model-supported limits and long inputs are not uniformly exploitable by current LLMs. We introduce R_FOLD, a bi-level context-management framework that jointly regulates observation admission and trajectory retention. READ processes newly returned observations at chunk granularity, preserving exact evidence when needed, distilling useful content, and discarding low-value material before it enters persistent context. FOLD compresses selected historical steps into continuation-oriented summaries over a step-addressable trajectory state. On BrowseComp-Plus, across three model backends and two context budgets, R_FOLD consistently outperforms ReAct and reactive summarization baselines, reaching 81.33% accuracy and showing strong robustness under a 32,768-token budget. These results demonstrate the value of bi-level context management as a practical paradigm for constructing compact, decision-relevant agent state under constrained serving budgets.

1 Introduction

Large language model (LLM) agents increasingly solve real-world tasks by repeatedly invoking tools, inspecting returned observations, and revising their actions over long interaction horizons, as in ReAct-style tool use and long-horizon web or research agents (Yao et al., 2022; Chen et al., 2025; Ye et al., 2025). The difficulty is that tool outputs are often verbose, unevenly relevant, and noisy: web-search observations may contain many truncated pages, full-page visits can return thousands of words, and multi-step trajectories can accumulate tens of thousands of input tokens in only a few turns (Chen et al., 2025; Lindenbauer et al., 2025; Kang et al., 2025). Directly appending this material to the active context can consume the context budget and make later decisions depend on low-value or distracting information, a problem related to broader findings that long inputs are not uniformly usable and that evidence in the middle of long contexts can be missed (Liu et al., 2024; Du et al., 2025). The systems constraint is equally important: Table 1 shows that publicly documented self-hosted profiles often provision service windows far below the declared context capacity, especially when deployment recipes optimize for accelerator memory, latency, throughput, precision format, or multimodal workloads. Thus, long-horizon agents can produce observation-heavy trajectories whose token demand is poorly matched to practical serving budgets. The central question is not simply how much history an agent can retain, but what should enter its context and in what representation.

Existing agent context-management approaches commonly focus on cross-step compression after interaction history has accumulated. Folding-style agents compress completed trajectory segments into summaries (Ye et al., 2025; Sun et al., 2025; Shao et al., 2025), summary-augmented training and inference methods learn or trigger periodic history condensation (Wu et al., 2025; Lu et al., 2025), and memory-centric agents maintain evolving progress briefs or external state across long tasks (Wan et al., 2025; Zhou et al., 2025; Liu et al., 2025; Zhang et al., 2026b). These approaches reduce historical volume and help agents continue beyond fixed-window limits, but they largely assume that observations have already been converted into useful state before being compressed. In long-horizon browsing and research tasks, this assumption is fragile: verbose retrieved documents and tool logs can pollute the active context immediately after they are returned, before any later trajectory-level summary is triggered (Lindenbauer

Table 1: Declared context capacity versus provisioned context length in publicly documented self-hosted serving profiles for frontier open-weight models.

Model	Declared	Provisioned	Retained	Documented self-hosted profile
`DeepSeek-V4-Flash`	1,048,576	135,168	12.9%	Atlas 800 A2, 8×64G, memory-bound serving
`MiniMax-M2.5`	196,608	32,768	16.7%	Atlas 800 A3, short-context low-latency profile
`Kimi-K2.5`	262,144	32,768	12.5%	Atlas 800 A3, DP4+TP4 high-throughput profile
`Qwen3-Coder-480B`	262,144	32,000	12.2%	vLLM, BF16, 8×H200/H20 recipe
`Qwen3-VL-235B`	262,144	128,000	48.8%	vLLM, 8×A100/H100 multimodal profile

Notes. Declared context is the documented model capacity; provisioned context is the cited self-hosted `max-model-len`, not a universal limit. Retained fraction is provisioned context divided by declared context. Sources: vLLM Recipes (2026a); vLLM Ascend (2026a); vLLM Recipes (2026b); vLLM Ascend (2026c); vLLM Recipes (2026c); vLLM Ascend (2026b); Qwen (2025); vLLM Recipes (2026d,e).

et al., 2025; Kang et al., 2025; Wang et al., 2026). Cross-step compression therefore addresses only part of the problem: robust context management also needs to decide, within each step, which portions of a newly obtained observation deserve admission into persistent context.

In this paper, we propose R_FOLD, a bi-level context-management framework for long-horizon tool-using agents under bounded context budgets. The design is motivated by how humans handle complex research materials: a researcher reading a long search result or document does not copy every passage into working notes, but inspects the material, preserves exact evidence when needed, records useful intermediate conclusions, and discards irrelevant detail before continuing. R_FOLD operationalizes this pattern through a step-addressable context representation and two complementary actions. READ partitions a lengthy tool observation into chunks and decides whether each chunk should be preserved verbatim, distilled into a compact note, or dropped before entering persistent context. FOLD compresses selected ranges of historical steps into structured handoff summaries that record established findings, unresolved leads, failed paths, and continuation state. Together, the two operations control both observation admission and trajectory retention, constructing a compact context that is useful for the agent's next decisions.

We evaluate R_FOLD on BrowseComp-Plus (Chen et al., 2025), a controlled long-horizon browsing benchmark that requires iterative search, page inspection, evidence tracking, and answer synthesis. Across `DeepSeek-V4-Flash` (DeepSeek-AI, 2026), `GPT-5.4-nano` (OpenAI, 2026), and `MiniMax-M2.5` (MiniMax, 2026), and under 32,768- and 65,536-token budgets, R_FOLD consistently outperforms vanilla ReAct and a reactive summary agent. The gains are largest under the tighter 32,768-token budget: R_FOLD reaches 81.33% accuracy with `DeepSeek-V4-Flash`, 54.67% with `GPT-5.4-nano`, and 70.00% with `MiniMax-M2.5`. These results show that proactive observation reading and step-level folding improve not only task accuracy, but also robustness to practical context constraints.

In summary, this paper makes the following contributions:

- **Bi-level context management.** We introduce R_FOLD, which jointly manages step-level observation ingestion and trajectory-level memory compression through READ and FOLD.
- **Step-addressable agent state.** We represent history as ordered action-observation steps, enabling fine-grained reading of new observations and structured folding of prior progress.
- **Long-horizon benchmark performance.** We show consistent gains on BrowseComp-Plus across three model backends and two context budgets.

2 Related Work

2.1 Trajectory-Level Context and Resource Management

Long-horizon tool use strains the trajectory state maintained by ReAct-style agents, whose interleaved thoughts, actions, and observations accumulate in an append-only manner over time (Yao et al., 2022). Prior work addresses this bottleneck along three related directions. A first direction makes context management an explicit agent behavior, often learned from trajectories or optimized through reinforcement learning. AgentFold, Context-Folding/FoldGRPO, FoldAct, ReSum, and

CAT train or prompt agents to fold, summarize, or otherwise compact completed trajectory segments into reusable progress states (Ye et al., 2025; Sun et al., 2025; Shao et al., 2025; Wu et al., 2025; Liu et al., 2025). These methods are closest to our work in treating context management as part of the agent loop, but their compression primarily operates after a sequence of actions and observations has already accumulated.

A second direction externalizes state into memory or workspace objects so that not all progress must remain in the main prompt. MEM1 learns a compact shared state, MemAct treats memory editing as an action, A-MEM dynamically links memory notes, and related memory systems organize experience into retrievable long-term structures (Zhou et al., 2025; Zhang et al., 2026b; Xu et al., 2025; Zhang et al., 2026a). These systems emphasize persistent memory organization, whereas R_FOLD focuses on the active context used during a single tool-use trajectory.

A third direction uses multi-agent or workspace-centered coordination to separate task execution from context organization. COMPASS assigns context maintenance to specialized components, and Git Context Controller organizes agent context through versioned workspace operations that support recovery and reuse across sessions or trajectories (Wan et al., 2025; Wu et al., 2026). These approaches reduce the burden of accumulated history, but they generally intervene after observations have entered some form of agent state. R_FOLD is complementary: it first controls which observations are admitted and in what form, and only then folds historical steps when the retained trajectory itself becomes too large.

2.2 Observation Compression and Agent State Construction

Observation-level work asks a different question: how should a tool return or retrieved document be transformed before it becomes part of the agent state? In long-document reading, ReadAgent, MemAgent, and Chain of Agents avoid feeding an entire document into a single monolithic context; instead they segment input, maintain local or episodic memory, or pass compact intermediate representations across readers (Lee et al., 2024; Yu et al., 2025; Zhang et al., 2024).

In retrieval and context-compression settings, Selective Context, CompAct, and Provence prune or compress retrieved text, observation masking and SWE-Pruner show that targeted omission of low-value observations can yield strong efficiency-performance tradeoffs in agent settings, and ACON further optimizes condensations of both environment observations and interaction histories (Li et al., 2023; Yoon et al., 2024; Chirkova et al., 2025; Lindenbauer et al., 2025; Wang et al., 2026; Kang et al., 2025).

Related memory-centric retrieval systems such as HippoRAG suggest that effectiveness often comes not from shortening text alone, but from reorganizing evidence into a more retrievable and relational representation (Gutiérrez et al., 2025). The common lesson is that useful compression must preserve answer-critical details, unresolved leads, and source relationships rather than merely reduce token count. R_FOLD brings this principle into the agent loop by making reading an explicit action conditioned on the current query, prior steps, and triggering tool call, so each observation is converted into task-oriented state before trajectory-level folding is applied.

3 Methodology

3.1 Overview

We formulate context management in tool-use agents as a step-wise R_FOLD process over long interaction trajectories. Rather than maintaining the active context as an append-only ReAct trace, the framework represents history as a structured list of agent steps. A newly returned observation is selectively processed before entering this list, and contiguous historical ranges can later be folded into compact handoff summaries.

At agent step t, the agent receives a user query x, has access to tool schemas S, and maintains a step-organized history H_t. The active context supplied to the language model is

$$C_t = \langle x, S, H_t \rangle. \tag{1}$$

The history is represented as an ordered list of items indexed by the agent steps whose retained content they cover:

$$H_t = \langle h_1^{(a_1:b_1)}, h_2^{(a_2:b_2)}, \ldots, h_{K_t}^{(a_{K_t}:b_{K_t})} \rangle. \tag{2}$$

An item $h_j^{(a_j:b_j)}$ represents one retained step item when $a_j = b_j$, and a folded summary of a contiguous step interval when $a_j < b_j$. Here j

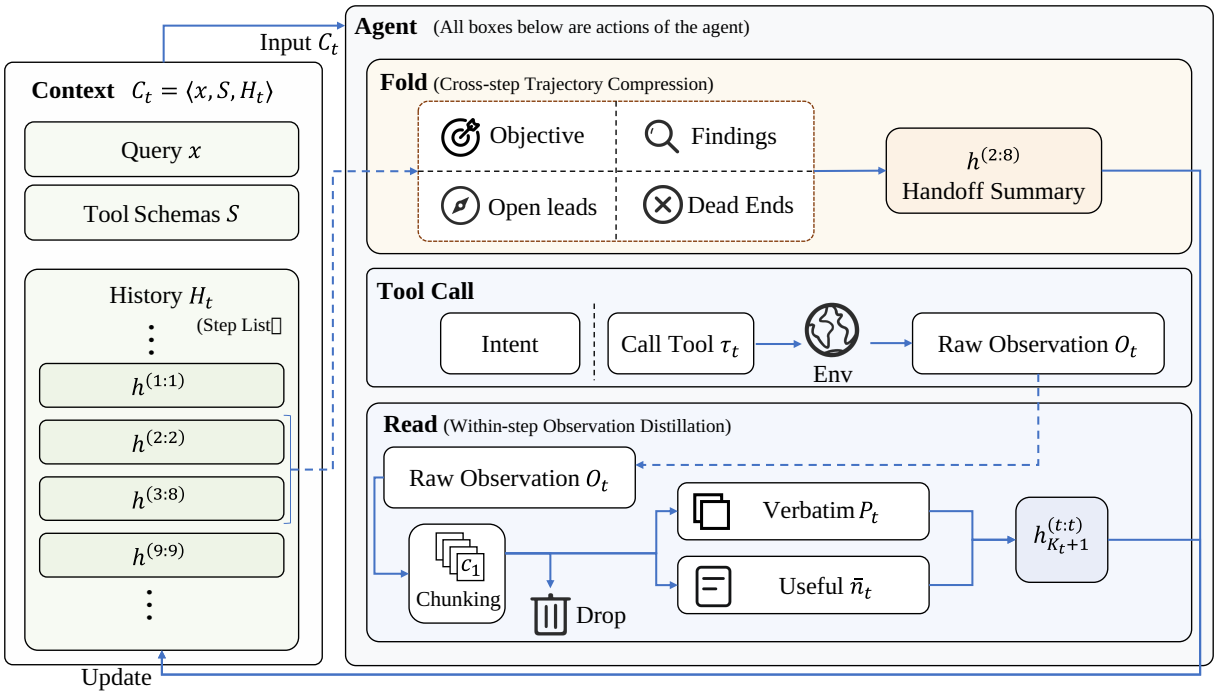

Figure 1: Overview of R_FOLD. A newly returned tool observation is processed through chunk-based reading before it is stored as a step item; selected historical step ranges are subsequently compressed through interval-level folding.

indexes the current stored item in the list, while $(a_j : b_j)$ records the agent steps covered by that item. For example, a trajectory may be stored as

$$\langle h^{(1:1)}, h^{(2:2)}, h^{(3:8)}, h^{(9:9)}, h^{(10:13)} \rangle. \tag{3}$$

This representation preserves chronological structure while allowing completed or less active segments to be replaced by compact state descriptions. The framework contains two complementary context-management operations. *Observation reading* determines which parts of a newly returned observation are retained verbatim, summarized, or discarded before the new step is stored. *Trajectory folding* replaces selected historical intervals with handoff summaries that preserve their local objective, key findings, and remaining state.

3.2 Step-List Context and Fold Action

Each individual step item stores a tool invocation together with the processed observation content retained for subsequent reasoning. Fold is itself an agent action: at step t, the agent may choose to replace a selected interval of the current step list with a compact handoff summary. Given a selected contiguous interval $r_t = [\ell, u]$ of current list positions, the corresponding covered step span is $(a_\ell : b_u)$. A folded handoff summary is generated as

$$z_t^{(a_\ell:b_u)} = \mathcal{F}(H_t[\ell : u] \mid C_t), \tag{4}$$

where $z_t^{(a_\ell:b_u)}$ records the objective pursued in the interval, the evidence or intermediate results obtained, unresolved assumptions or open leads, and dead ends that should not be revisited. The interval is then replaced by the generated summary:

$$\begin{aligned}\mathrm{Fold}(H_t; [\ell, u]) = \langle & h_1^{(a_1:b_1)}, \ldots, \\ & h_{\ell-1}^{(a_{\ell-1}:b_{\ell-1})}, z_t^{(a_\ell:b_u)}, \\ & h_{u+1}^{(a_{u+1}:b_{u+1})}, \ldots, h_{K_t}^{(a_{K_t}:b_{K_t})} \rangle.\end{aligned} \tag{5}$$

Fold can be invoked through either of two triggers. First, the main agent may initiate folding when it identifies a locally completed subtask, an exhausted search branch, or a coherent intermediate state that can be handed off compactly. Second, folding is triggered when the active context approaches a predefined fraction of the maximum context budget. In both cases, the model selects the interval r_t to compress; the budget trigger only determines that a Fold action should be taken. Let $W_{\max}$ denote the available context-window budget in tokens, $|C_t|$ the current token count, and

$\rho \in (0,1)$ a preset utilization threshold. The fold-trigger indicator for step t is

$$\gamma_t = \max\left\{\gamma_t^{\text{agent}}, \mathbb{I}[|C_t| \geq \rho W_{\max}]\right\}, \quad (6)$$

where $\gamma_t, \gamma_t^{\text{agent}} \in \{0,1\}$, and $\gamma_t^{\text{agent}} = 1$ denotes an agent-initiated Fold action. Thus, folding is selective with respect to the interval being compressed, while the decision to invoke it can be either proactive or budget-driven. This mechanism retains fine-grained details for active regions of the trajectory and compresses regions whose contribution can be expressed as task-oriented handoff state.

When a Fold action is executed at step t, it advances the trajectory by replacing the selected interval in the stored step list. The resulting next-step history and active context are defined together with the tool-use update rule in Section 3.4.

3.3 Chunk-Based Observation Reading

Tool observations, including webpages, retrieved documents, execution logs, source files, and search results, often mix task-critical evidence with irrelevant or redundant material. Directly appending a complete observation would cause low-value content to consume the context budget. We therefore process each newly returned observation before constructing the corresponding step item.

Given a tool invocation τ_t and its returned observation O_t at step t, the system first segments the observation into sequential chunks:

$$O_t = \langle c_1, c_2, \ldots, c_{L_t} \rangle. \quad (7)$$

We use the same model backend as the main agent to execute the READ action over different chunks, implemented as separate instruction-prompted calls. For each chunk c_i, the agent receives the current context C_t, the tool invocation τ_t, and the chunk content, then produces a compact note and an information-value label:

$$(n_i, v_i) = \mathcal{R}(c_i \mid C_t, \tau_t). \quad (8)$$

After processing each chunk, the agent judges the value of that chunk for the continuing task. The label space is

$$v_i \in \mathcal{V} = \{\mathsf{verbatim}, \mathsf{useful}, \mathsf{drop}\}. \quad (9)$$

A chunk labeled verbatim contains information for which exact fidelity is important, such as source code, formal requirements, quoted evidence, structured outputs, or numerical results. A chunk labeled useful contributes task-relevant information that can be retained as a compact note. A chunk labeled drop is irrelevant, redundant, or otherwise unnecessary for the current task and is omitted from the stored step representation. The prompt asks the agent to judge each chunk by its expected value for the current objective, unresolved leads, and downstream answer synthesis, while preserving exact text only when paraphrasing could change the answer-relevant content.

The preserved chunks and useful notes are collected as order-preserving subsequences, with indices taken in increasing order:

$$\begin{aligned} I_t^{\mathsf{ver}} &= \{i \mid v_i = \mathsf{verbatim}\}, \\ I_t^{\mathsf{use}} &= \{i \mid v_i = \mathsf{useful}\}, \\ P_t &= \langle c_i \rangle_{i \in I_t^{\mathsf{ver}}}, \\ N_t &= \langle n_i \rangle_{i \in I_t^{\mathsf{ver}} \cup I_t^{\mathsf{use}}}. \end{aligned} \quad (10)$$

A separate aggregation operator then integrates the retained information into a step-level handoff note:

$$\bar{n}_t = \mathcal{R}_{\text{agg}}(C_t, N_t). \quad (11)$$

The distinction between $\mathcal{R}$ and $\mathcal{R}_{\text{agg}}$ makes the two stages explicit: the first performs local relevance and fidelity decisions, while the second produces a coherent observation-level record. Both operators use the same model backend but are invoked with different instructions: $\mathcal{R}$ is a chunk-level classification-and-note prompt, whereas $\mathcal{R}_{\text{agg}}$ is a separate aggregation prompt over the retained notes. Preserved chunks remain available in their exact form; the aggregate note provides a compact description of their role together with the information summarized from useful chunks.

3.4 R_FOLD Context Update

The context update depends on the type of agent step being executed. For a tool-use step t, observation reading produces the stored content that will be appended to the step list. After reading, the tool invocation and its processed result form one new individual step item:

$$h_{K_t+1}^{(t:t)} = \langle \tau_t, P_t, \bar{n}_t \rangle. \quad (12)$$

Let a_t denote the action type at step t, with $a_t =$ tool for an external tool-use step and $a_t =$ fold for a Fold step. The next history is

$$H_{t+1} = \begin{cases} \mathrm{Append}\left(H_t, h_{K_t+1}^{(t:t)}\right), & a_t = \mathsf{tool}, \\ \mathrm{Fold}(H_t; r_t), & a_t = \mathsf{fold}. \end{cases} \tag{13}$$

The active context for the next model invocation is then

$$C_{t+1} = \langle x, S, H_{t+1} \rangle. \tag{14}$$

This unified update rule makes the relationship between the two operations explicit. A tool-use step appends a processed observation item, while a Fold step replaces selected stored ranges with a handoff summary. Because both operations act on the same step-organized history and both produce the next-step context C_{t+1}, the framework preserves exact information when needed, maintains task-oriented handoff state across long trajectories, and avoids repeatedly attending to low-value or already-resolved details.

4 Experiments

We evaluate the proposed R_FOLD framework from three perspectives: effectiveness on long-horizon agentic tasks, the individual and combined contributions of the *Read* and *Fold* components, and efficiency under constrained context budgets. Our evaluation is centered on resource-constrained settings, in which the agent needs to preserve a useful reasoning state without continuously expanding its active prompt with verbose tool observations.

4.1 Experimental Setup

We evaluate on BrowseComp-Plus (Chen et al., 2025), a controlled browsing benchmark over a fixed corpus, using `Qwen3-Embedding-8B` as the retriever for all runs. Following the Fold-GRPO evaluation setting (Sun et al., 2025), we use a 150-question subset with 50 easy, 50 normal, and 50 hard examples. The benchmark provides SEARCH, which returns truncated retrieved pages, and VISIT, which returns a full page document. We test three efficiency-oriented backends: `DeepSeek-V4-Flash`, `GPT-5.4-nano`, and `MiniMax-M2.5`. The baselines are vanilla ReAct, which appends the raw trajectory until the context budget is reached, and Summary Agent (Wu et al., 2025), which reactively summarizes the accumulated history at the same threshold used by R_FOLD. All methods use identical task prompts, tool interfaces, decoding settings, and retriever configuration. The main comparison uses 32,768- and 65,536-token context budgets, a 100-tool-call limit, and a 4096-token output limit. For R_FOLD, READ uses the same backend model as the main agent, with 700-word chunks, and budget-triggered folding is activated at $0.85W_{\max}$, where $W_{\max}$ is the active context budget. We report official answer accuracy, average tool calls, SEARCH/VISIT calls, total input-token consumption, KV-cache hit rate, and average input length.

4.2 Main Results

Table 2 reports both answer accuracy and tool-use behavior on BrowseComp-Plus for all three model backends under the 32,768- and 65,536-token budgets.

R_FOLD is consistently more accurate. R_FOLD obtains the best accuracy in every model and budget setting. Under the 32,768-token budget, it reaches 81.33% with `DeepSeek-V4-Flash`, 54.67% with `GPT-5.4-nano`, and 70.00% with `MiniMax-M2.5`, substantially outperforming both ReAct and Summary Agent. These results indicate that the overall bi-level context-management framework is effective when the active context is strongly constrained.

R_FOLD actively enables deeper exploration. With `DeepSeek-V4-Flash`, R_FOLD sustains prolonged trajectories, performing roughly 22 tool calls per task and around 18 searches, far more than ReAct under the same budget. ReAct terminates after very shallow exploration at 32,768 tokens, averaging only 2.1, 1.8, and 1.7 tool calls across the three backends. Summary Agent can continue beyond the point where ReAct would exhaust the context because it compresses the accumulated history when the threshold is reached. However, its compression is reactive and session-level: it summarizes the whole carried context rather than reorganizing selected trajectory ranges. In contrast, R_FOLD can fold specific historical intervals and can invoke folding before the context is full, allowing the agent to preserve the local reasoning state while continuing exploration.

R_FOLD preserves performance under tight budgets. The 32,768-token setting is the more restrictive condition, yet R_FOLD already recovers most of its 65,536-token performance there. Its accuracy changes by 0.00, 2.66, and 6.00 points when the budget is doubled for `DeepSeek-V4-Flash`, `GPT-5.4-nano`, and `MiniMax-M2.5`, respectively. This pattern suggests that the framework helps the

Table 2: Main results on BrowseComp-Plus under two context budgets. Accuracy is the official task metric; interaction statistics are averages per task.

Model	Method	Context budget	Acc. (%)	Tools	Search	Visit
DeepSeek-V4-Flash	ReAct	32,768	4.67	2.1	1.8	0.3
DeepSeek-V4-Flash	Summary Agent	32,768	18.67	4.6	3.6	1.0
DeepSeek-V4-Flash	R_FOLD	32,768	**81.33**	**21.9**	**17.9**	**4.0**
DeepSeek-V4-Flash	ReAct	65,536	21.33	3.5	2.7	0.8
DeepSeek-V4-Flash	Summary Agent	65,536	63.33	15.9	12.5	3.4
DeepSeek-V4-Flash	R_FOLD	65,536	**81.33**	**22.2**	**18.2**	**4.0**
GPT-5.4-nano	ReAct	32,768	27.33	1.8	1.7	0.2
GPT-5.4-nano	Summary Agent	32,768	48.67	3.5	3.2	**0.3**
GPT-5.4-nano	R_FOLD	32,768	**54.67**	**4.3**	**4.1**	0.2
GPT-5.4-nano	ReAct	65,536	41.33	2.1	2.0	0.1
GPT-5.4-nano	Summary Agent	65,536	48.67	3.5	3.3	**0.2**
GPT-5.4-nano	R_FOLD	65,536	**57.33**	**4.0**	**3.9**	**0.2**
MiniMax-M2.5	ReAct	32,768	4.67	1.7	1.6	0.1
MiniMax-M2.5	Summary Agent	32,768	12.67	2.2	2.0	0.1
MiniMax-M2.5	R_FOLD	32,768	**70.00**	**10.3**	**9.9**	**0.4**
MiniMax-M2.5	ReAct	65,536	24.67	2.8	2.5	0.3
MiniMax-M2.5	Summary Agent	65,536	58.00	4.6	3.8	**0.8**
MiniMax-M2.5	R_FOLD	65,536	**76.00**	**10.1**	**9.7**	0.4

underlying model use a small active context more effectively, rather than relying primarily on a larger raw history.

4.3 Ablation Studies

4.3.1 Read vs. Fold

We study the contributions of the two core components by starting from the complete R_FOLD framework and removing modules. All ablation experiments use MiniMax-M2.5 with a fixed context budget of 65,536 tokens. Table 3 compares the full method against R_FOLD without FOLD, without READ, and without both modules. The last setting retains only the step-organized context representation, allowing us to examine whether the representation alone explains the gains or whether the active context-management actions are necessary. The w/o-both variant reaches 21.33%, showing that the step-organized representation alone does not explain the gains. Without READ and FOLD, the agent still has to carry verbose tool observations in the active context, so the step list by itself cannot reliably expose its advantage under the current budget. This result is consistent with the role of the representation: it is designed to support long-horizon, multi-step state organization, but it needs observation reading and trajectory folding to keep that state compact and usable. Adding active context-management operations raises accuracy to 76.00%. READ provides the larger individual contribution: the no-FOLD variant, which keeps READ, reaches 70.67%, whereas disabling READ lowers accuracy to 58.67%. FOLD remains complementary, improving performance from the no-FOLD variant to the full system by 5.33 points.

Table 3: Ablation study on MiniMax-M2.5 with a 65,536-token context budget.

Method	Accuracy (%)
R_FOLD	**76.00**
R_FOLD w/o FOLD	70.67
R_FOLD w/o READ	58.67
R_FOLD w/o both	21.33

4.4 Efficiency Analysis

Table 4 presents efficiency statistics for DeepSeek-V4-Flash under both context budgets. The results provide a complementary view to task accuracy and token-level cost. R_FOLD exhibits higher aggregate input-token consumption, which stems from the auxiliary coordination calls required by READ and FOLD to support extended trajectories. At the same time, the measured average input length suggests a potential serving-side benefit: R_FOLD exchanges auxiliary prefill work for a more compact active state during the main reasoning loop.

Context budget sensitivity is reduced. Figure 2 isolates the effect of the active context budget on BrowseComp-Plus accuracy. R_FOLD is the least sensitive to the smaller budget among the compared methods. Even at 32,768 tokens, it reaches 100.0%, 95.4%, and 92.1% of its own 65,536-token accuracy for DeepSeek-V4-Flash, GPT-5.4-nano, and MiniMax-M2.5, respectively. This indicates that R_FOLD can preserve much of the model's task performance under a compressed active context, consistent with its goal of keeping compact decision-relevant state rather than long raw traces.

Summary Agent also consumes more input to-

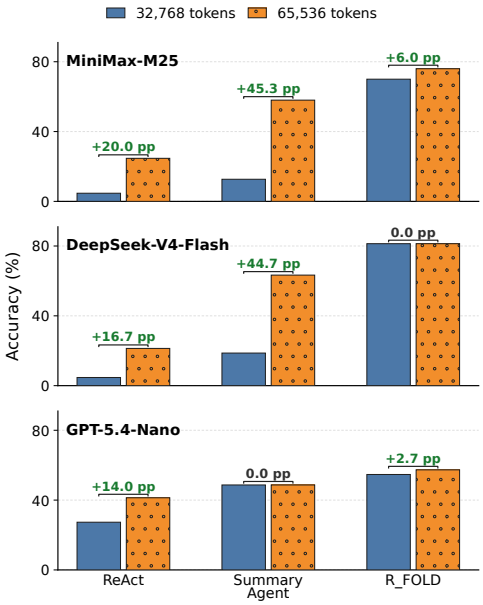

Figure 2: Absolute BrowseComp-Plus accuracy change when increasing the active context budget from 32,768 to 65,536 tokens.

kens than ReAct because summarization permits additional interaction after the raw trajectory approaches the context limit. In contrast, plain ReAct often reaches the budget earlier and therefore terminates after a shallower exploration trajectory. R_FOLD further increases total input tokens because chunk-level reading and folding introduce auxiliary calls, but this cost supports substantially longer exploration and higher final accuracy. In this sense, total token count reflects an explicit accuracy–compute trade-off rather than a pure serving-cost metric.

The average input lengths show a potential systems implication of R_FOLD: despite its larger total token consumption, each individual reasoning call is much shorter than the corresponding Summary Agent prompt, especially under the 65,536-token budget. In principle, shorter per-call inputs can reduce prefill work and KV-cache pressure per active request, but we do not measure end-to-end latency, first-token latency, GPU memory, throughput, concurrency behavior, or the wall-clock overhead of auxiliary READ/FOLD calls. Overall, R_FOLD trades secondary context-management overhead for a compact active state that enables deeper exploration and prevents reasoning degradation; measuring the resulting serving behavior is left for future work.

Table 4: Efficiency analysis on `DeepSeek-V4-Flash` under two context budgets. Tool calls are averages per task; input-token totals and input tokens per correct answer are reported in millions (M).

Method	Context budget	Tools	Input Tok.	Tok./Correct	KV Hit (%)	Avg. Input
ReAct	32,768	2.1	3.6M	0.51M	34.20	10984
Summary Agent	32,768	4.6	172.7M	6.17M	**87.67**	13835
R_FOLD	32,768	**21.9**	**598.6M**	4.91M	72.19	**6202**
ReAct	65,536	3.5	15.9M	0.50M	49.76	27619
Summary Agent	65,536	15.9	296.9M	3.13M	62.92	43375
R_FOLD	65,536	**22.2**	**623.7M**	5.11M	**72.96**	**6311**

5 Conclusion

We presented R_FOLD, a proactive context-management framework for long-horizon tool-using agents. The framework separates two sources of context pressure: noisy incoming observations and accumulated historical trajectories. Read filters and distills observations before they enter persistent context, while Fold compresses selected ranges of historical steps into handoff-oriented summaries. Together, these operations convert raw tool-use traces into a compact, structured task state that is intended to support more stable long-horizon reasoning under finite attention and inference budgets.

Limitations

R_FOLD adds model calls for chunk-level reading and trajectory folding, so latency- or cost-sensitive deployments may need adaptive policies for deciding when these operations are worthwhile. The evaluation is currently centered on BrowseComp-Plus, a text-heavy browsing benchmark; results on software-engineering, data-analysis, multimodal, or highly interactive environments remain to be established. Finally, R_FOLD relies on model judgments about whether chunks should be preserved, summarized, or dropped, so mistaken read decisions can still remove weak clues that become important later.

References

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 [cs].

Nadezhda Chirkova, Thibault Formal, Vassilina Nikoulina, and Stéphane Clinchant. 2025.

Provence: efficient and robust context pruning for retrieval-augmented generation. *arXiv preprint*. ArXiv:2501.16214 [cs].

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A. Huerta, and Hao Peng. 2025. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. *arXiv preprint*. ArXiv:2510.05381 [cs].

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2025. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint*. ArXiv:2405.14831 [cs].

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. *arXiv preprint*. ArXiv:2510.00615.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint*. ArXiv:2402.09727.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing Context to Enhance Inference Efficiency of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353, Singapore. Association for Computational Linguistics.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. 2025. The complexity trap: Simple observation masking is as efficient as LLM summarization for agent context management. DL4C at NeurIPS 2025 poster.

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.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. 2025. Context as a Tool: Context Management for Long-Horizon SWE-Agents. *arXiv preprint*. ArXiv:2512.22087 [cs].

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 [cs].

MiniMax. 2026. MiniMax M2.5 model release.

OpenAI. 2026. GPT-5.4 nano Model | OpenAI API.

Qwen. 2025. Qwen3-Coder-480B-A35B-Instruct. Official model card.

Jiaqi Shao, Yufeng Miao, Wei Zhang, and Bing Luo. 2025. FoldAct: Efficient and Stable Context Folding for Long-Horizon Search Agents. *arXiv preprint*. ArXiv:2512.22733 [cs].

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 [cs].

vLLM Ascend. 2026a. DeepSeek-V4-Flash. Deployment guide.

vLLM Ascend. 2026b. Kimi-K2.5. Deployment guide.

vLLM Ascend. 2026c. MiniMax-M2.5. Deployment guide.

vLLM Recipes. 2026a. DeepSeek-V4-Flash. Deployment recipe.

vLLM Recipes. 2026b. MiniMaxAI/MiniMax-M2.5. Deployment recipe.

vLLM Recipes. 2026c. moonshotai/Kimi-K2.5. Deployment recipe.

vLLM Recipes. 2026d. Qwen3-Coder-480B-A35B-Instruct usage guide. Deployment guide.

vLLM Recipes. 2026e. Qwen3-VL-235B-A22B-Instruct. Deployment guide.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. 2025. COMPASS: Enhancing Agent Long-Horizon Reasoning with Evolving Context. *arXiv preprint*. ArXiv:2510.08790 [cs].

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. 2026. SWE-Pruner: Self-Adaptive Context Pruning for Coding Agents. *arXiv preprint*. ArXiv:2601.16746 [cs].

Junde Wu, Minhao Hu, Jiayuan Zhu, Jiazhen Pan, Yuyuan Liu, Min Xu, and Yueming Jin. 2026. Git Context Controller: Manage the Context of LLM-based Agents like Git. *arXiv preprint*. ArXiv:2508.00031 [cs] version: 2.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. 2025. ReSum: Unlocking Long-Horizon Search Intelligence via Context Summarization. *arXiv preprint*. ArXiv:2509.13313 [cs].

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 [cs].

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. 2025. AgentFold: Long-Horizon Web Agents with Proactive Context Management. *arXiv preprint*. ArXiv:2510.24699 [cs].

Chanwoong Yoon, Taewhoo Lee, Hyeon Hwang, Minbyul Jeong, and Jaewoo Kang. 2024. CompAct: Compressing Retrieved Documents Actively for Question Answering. *arXiv preprint*. ArXiv:2407.09014 [cs].

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. *arXiv preprint*. ArXiv:2507.02259 [cs].

Ningning Zhang, Xingxing Yang, Zhizhong Tan, Weiping Deng, and Wenyong Wang. 2026a. HiMem: Hierarchical Long-Term Memory for LLM Long-Horizon Agents. *arXiv preprint*. ArXiv:2601.06377 [cs].

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. 2024. Chain of Agents: Large Language Models Collaborating on Long-Context Tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. 2026b. Memory as Action: Autonomous Context Curation for Long-Horizon Agentic Tasks. *arXiv preprint*. ArXiv:2510.12635 [cs].

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. *arXiv preprint*. ArXiv:2506.15841 [cs].

A R_FOLD Prompts and Context Organization

This appendix gives the implementation-level prompts, state format, read-mode observation construction, tool schemas, and baseline context organization used in our experiments.

A.1 R_FOLD Main Prompt and Step List

System prompt. R_FOLD uses the following system prompt in the main decision loop. The available function descriptions are appended after this prompt using the same schema renderer for all methods.

```
You are an agent with access to
tools. Use the available tools
according to their descriptions
and parameter schemas to answer
the user's question. Manage any
provided context as needed.
When you have enough information
to answer, call the submit_answer
tool with the final answer.
```

The user message has two parts: the original task and the current step list.

```
## Query
{user question}

## Step List
{rendered step list}
```

Step-list representation. The persistent trajectory is not represented as a raw chat transcript. It is rendered as an ordered list of step records. A normal external-tool step is rendered as follows:

```
**[Step i]**
Thought:
{assistant text before the tool call}

Action call:
{"name": "...", "arguments": {...}}

Action result:
{processed observation}
```

When R_FOLD executes a FOLD action, the selected range is replaced by a single folded step:

```
**[Step a to b]**
Local Objective: ...
Retained Findings: ...
Handoff State: ...
```

During ordinary decision making, the model can call task tools, `submit_answer`, or `fold`. During READ, the model is instead placed in a transient read step, denoted `[Step i(Reading)]`, and the only available tool is `read_action`. This separation ensures that raw observations are processed before they are admitted into the persistent step list.

A.2 Read-Mode Prompting and Chunk Organization

Chunk-level read prompt. For a raw observation chunk, R_FOLD sends the original query, the

current step list, and one active read step. The active read step contains the triggering tool call, the chunk index, the current observation block, and the read-mode instruction:

```
## Active Read Step
**[Step i(Reading)]**
**Action call:**
{"name": "visit",
 "arguments": {...}}
**Chunk:** k/n

Current obs:
[Chunk k]
{decorated chunk text}

You are currently in READ MODE.
The active reading step is
[Step i(Reading)] for action visit.
The current observation block is
chunk k/n. You are reading raw
content that may later be merged
with notes from other chunks.
You may only call the read_action
tool for this turn. Write the note
with `Local Evidence`,
`Carry Forward Details`, and
`Open Status`. `Carry Forward
Details` should explicitly preserve
exact names, dates, titles, quotes,
caveats, or source relationships
that must survive into the merge
step. Set content_priority to
`preserve_verbatim` only when the
original chunk text itself must be
preserved exactly. Set it to
`useful` for normal useful chunks,
and `no_value` if this chunk
contributes no useful evidence.
Choose content_priority from
`preserve_verbatim`, `useful`,
or `no_value`.
```

The chunk is constructed by word spans, with a 700-word target length and 100-word overlap in our experiments. The chunk text is not passed as an isolated raw block. If the chunk does not start at the beginning of the original observation, R_FOLD can prepend a short beginning excerpt and an omitted-character marker. If the chunk does not end at the original observation boundary, it can append an omitted-character marker and a short ending excerpt. This gives the reader local content while preserving global boundary cues.

```
[Chunk 2]
{beginning excerpt from the
original observation}
[omitted 18342 chars from the start]
{current chunk body}
[omitted 9710 chars from the end]
{ending excerpt from the
original observation}
```

The model then calls `read_action` with a structured note. The note is required to separate local evidence, details that must be carried forward, and unresolved status. Chunks marked `no_value` are ignored in the merge step; chunks marked `preserve_verbatim` are additionally retained as exact text in the final processed observation.

Merge read prompt. After chunk-level reading, R_FOLD runs a final read-mode merge over the retained note list. This merge turn is still represented as `[Step i(Reading)]`, but it does not contain a raw action-call block or a chunk index because the current observation is the note list rather than page text:

```
Current obs:
The following are chunk-level read
notes collected from earlier raw
chunks. Synthesize them into one
merged note without dropping
decision-critical facts.

[Chunk 1 Note]
{note from chunk 1}

[Chunk 2 Note]
{note from chunk 2}
```

The merge instruction asks for a note organized as `Merged Findings`, `Critical Details to Preserve`, and `Remaining Gaps or Next Check`. The merged note becomes the compact observation stored in the step list. If any chunks were labeled `preserve_verbatim`, their exact decorated chunk text is appended under `Important detail result`. The stored result therefore has the form:

```
Useful information extracted from
the visit action result:

Note:
{merged note}

Important detail result:
```

```
[Chunk 1]
{preserved chunk text}
....
[Chunk 3]
{preserved chunk text}
```

A.3 R_FOLD Tool Schemas

The schemas are passed to the model as OpenAI-compatible `tools` objects. The main loop exposes `search`, `visit`, `submit_answer`, and `fold`; read mode exposes only `read_action`. The following excerpt shows the JSON structure used in our BrowseComp-Plus runs, with descriptions shortened for space.

```
[
  {
    "type": "function",
    "function": {
      "name": "search",
      "description": "Run web search.",
      "parameters": {
        "type": "object",
        "properties": {
          "intent": {"type": "string"},
          "query": {
            "type": "array",
            "items": {"type": "string"}
          },
          "top_k": {
            "type": "integer",
            "minimum": 1,
            "default": 5
          }
        },
        "required": ["intent", "query"],
        "additionalProperties": false
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "visit",
      "description": "Retrieve pages.",
      "parameters": {
        "type": "object",
        "properties": {
          "intent": {"type": "string"},
          "url": {
            "type": "array",
            "items": {"type": "string"}
          },
          "goal": {"type": "string"}
        },
        "required": [
          "intent",
          "url",
          "goal"
        ],
        "additionalProperties": false
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "submit_answer",
      "parameters": {
        "type": "object",
        "properties": {
          "answer": {"type": "string"}
        },
        "required": ["answer"],
        "additionalProperties": false
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "fold",
      "parameters": {
        "type": "object",
        "properties": {
          "intent": {"type": "string"},
          "compress_range": {
            "type": "array",
            "items": {"type": "integer"}
          },
          "folded_summary": {
            "type": "string"
          }
        },
        "required": [
          "intent",
          "compress_range",
          "folded_summary"
        ],
        "additionalProperties": false
      }
    }
  }
```

```
]
```

In read mode, the only exposed tool is:

```
{
  "type": "function",
  "function": {
    "name": "read_action",
    "parameters": {
      "type": "object",
      "properties": {
        "intent": {"type": "string"},
        "note": {"type": "string"},
        "content_priority": {
          "type": "string",
          "enum": [
            "preserve_verbatim",
            "useful",
            "no_value"
          ]
        }
      },
      "required": [
        "intent",
        "note",
        "content_priority"
      ],
      "additionalProperties": false
    }
  }
}
```

The `read_action` note is instructed to contain `Local Evidence`, `Carry Forward Details`, and `Open Status` for chunk-level reading, or `Merged Findings`, `Critical Details to Preserve`, and `Remaining Gaps or Next Check` for merge reading. The `folded_summary` field is instructed to contain `Local Objective`, `Retained Findings`, and `Handoff State`.

A.4 Baseline Prompts and Context Organization

ReAct baseline. The ReAct baseline uses the same BrowseComp-Plus task tools and the same schema renderer, but it does not use READ or FOLD. Its system and user prompts are:

```
You are an agent with access to
tools. Use the available tools
according to their descriptions
and parameter schemas to answer the
user's question.
When the task is complete, call the
submit_answer tool with the final
answer.

Question: {Question}
```

Its context is the raw chat history. After each tool call, the assistant message contains optional assistant text plus a JSON `Tool call:` block, and the next user message contains the raw tool observation. If the provider reports a context-limit error or the configured input budget is exceeded, the run is counted as failed. Thus, ReAct retains no explicit step list, read note, or folded handoff state.

Summary Agent baseline. The Summary Agent baseline uses the same initial system and user prompts as ReAct. It also keeps a raw chat-history context within a session, but when the context window approaches the configured threshold or a context-limit error is encountered, it asks the model to produce a handover summary. The summary prompt is:

```
Your context window is getting full.
Produce a concise handover summary
that will serve as the only
carried-over task state for the
next session.

### // RESEARCH STATE HANDOVER //

1. Mission Objective
* Original Query:
  [state the user's verbatim query]
* Verification Checklist:
  - [STATUS] item 1
  - [STATUS] item 2

2. Key Findings
* Critical verified facts with sources.
* Discrepancies between sources, if any.

3. Tactical Plan
* Promising leads.
* Known dead ends.
* Immediate next action.

Put the summary inside:
<summary>
</summary>
```

After extracting the text inside `<summary>` and `</summary>`, the agent starts a new session with the

original system and task prompt plus the following continuation message:

```
For this question, you have already
made the following progress in a
previous session, summarized as
follows:

{summary}

Now continue working on it.
```

Therefore, Summary Agent differs from R_FOLD in two ways. First, it admits raw observations into the active chat history before summarizing them. Second, its compression is session-level and reactive, whereas R_FOLD performs observation-level reading before admission and step-range folding inside a persistent step list.