U-Fold: Dynamic Intent-Aware Context Folding for User-Centric Agents

Anonymous ACL submission

Abstract

Large language model (LLM)-based agents have been successfully deployed in many tool-augmented settings, but their scalability is fundamentally constrained by context length. Existing context-folding methods mitigate this issue by summarizing past interactions, yet they are typically designed for single-query or single-intent scenarios. In more realistic user-centric dialogues, we identify two major failure modes: (i) they irreversibly discard fine-grained constraints and intermediate facts that are crucial for later decisions, and (ii) their summaries fail to track evolving user intent, leading to omissions and erroneous actions. To address these limitations, we propose U-Fold, a dynamic context-folding framework tailored to user-centric tasks. U-Fold retains the full user–agent dialogue and tool-call history but, at each turn, uses two core components to produce an intent-aware, evolving dialogue summary and a compact, task-relevant tool log. Extensive experiments on τ-bench, τ^2-bench, VitaBench, and harder context-inflated settings show that U-Fold consistently outperforms ReAct (achieving a 71.4% win rate in long-context settings) and prior folding baselines (with improvements of up to 27.0%), particularly on long, noisy, multi-turn tasks. Our study demonstrates that U-Fold is a promising step toward transferring context-management techniques from single-query benchmarks to realistic user-centric applications.

1 Introduction

Large language model (LLM)-based agents have rapidly advanced in tool-augmented applications, from web navigation and software control to life-service assistants (Schick et al., 2023; Shen et al., 2023; Zheng et al., 2025; Zhou et al., 2025b; Wang et al., 2023a). A key capability is reasoning over long interaction histories—thoughts, tool calls, and feedback (Yao et al., 2022b; Wei et al., 2022; Guo et al., 2025; Jaech et al., 2024). However, naively feeding the entire history to the model leads to context explosion, which exceeds token budgets and overwhelms the model's reasoning ability. This has spurred work on context folding (Wu et al., 2025; Ye et al., 2025; Chen et al., 2025; Zhou et al., 2025c), which iteratively compresses working memory (Wu et al., 2025; Ye et al., 2025; Chen et al., 2025) to keep context compact while preserving task-critical information.

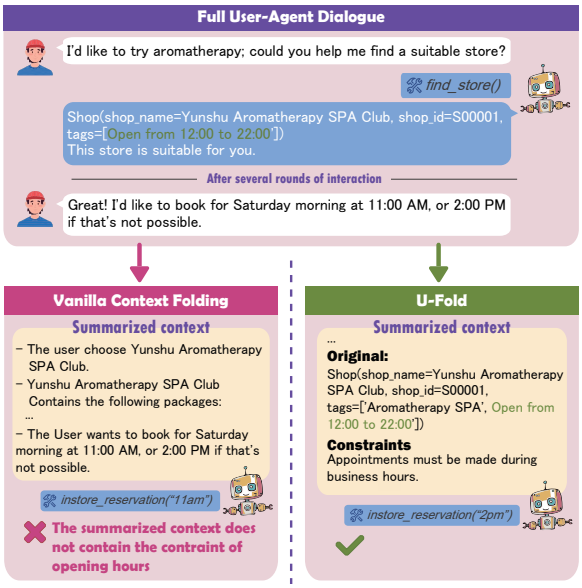

Figure 1: While vanilla context folding may drop critical constraints after multi-turn interactions and thus lead to incorrect actions, U-Fold preserves fine-grained information through dynamic context folding.

Although prior folding methods perform well on single-goal long-horizon benchmarks (Wei et al., 2025; Zhou et al., 2025a; Mialon et al., 2023; Shridhar et al., 2020; Yao et al., 2022a), their behavior in user-centric settings remains underexplored. In realistic practice, an agent interacts with a user over multiple turns, repeatedly uses tools, and must handle user's intent that changes over time (Yao et al., 2024; Barres et al., 2025; Qian et al., 2025a; He et al., 2025; Wang et al., 2023b; Lu et al., 2025). These

settings require higher accuracy and information completeness from folding. Through systematic observation, we find two key limitations of existing methods in user-centric environments. First, static summaries irreversibly drop fine-grained user constraints and intermediate facts (Figure 1) needed for correct tool use in later turns. Second, they do not track *shifting* user intent, often yielding summaries that lag behind or misrepresent the user's current needs (Figure 4).

To address these challenges, We propose **U-Fold**, a dynamic, intent-aware context-folding framework for user-centric agents. Instead of static summarization, U-Fold keeps the full user–agent history and, at each turn, builds a compact user-centric working context via two lightweight modules: (i) **Conversation Summarization**, which tracks dialogue evolution and maintains an up-to-date view of user intent; and (ii) **Dynamic Data Extraction**, which filters structured tool outputs to retain only fields relevant to current goals. The folded context is then used for reasoning and tool invocation. By aligning context construction with evolving intent, U-Fold reduces redundancy while preserving fine-grained constraints and intermediate facts essential for user-centric tasks.

We evaluate U-Fold on user-centric benchmarks (Yao et al., 2024; Barres et al., 2025; He et al., 2025) and find it consistently outperforms ReAct (Yao et al., 2022b) (71.4% win rate in long-context settings; Figure 3) and prior folding baselines (up to +27.0%). Our error analysis further highlights three dominant failure modes: (i) ***Miscomprehension of User Intent***, (ii) ***Omission of Critical User Information***, and (iii) ***Unrecognized User Errors***. U-Fold reduces all three by maintaining an intent-aligned, dynamically folded context (Section 4.4).

To summarize, our contributions are:

- We analyze existing context-folding strategies on realistic user-centric benchmarks and identify failures from intent drift and loss of user-specific constraints.
- We introduce U-Fold, which combines conversation summarization with dynamic data extraction to build a compact, informative working context that adapts to evolving user intent.
- We evaluate U-Fold across user-centric benchmarks, showing consistent gains over ReAct and prior folding baselines; ablations and error analyses underscore the value of dynamic user-centric folding for robust long-horizon behavior.

2 Related Work

2.1 User-Centric Agent

Recent research on intelligent agents has increasingly emphasized user-centric interaction (Yao et al., 2024; Barres et al., 2025; Qian et al., 2025a; He et al., 2025; Qian et al., 2025b; Wang et al., 2023b). Traditional evaluations focused on static tasks fail to capture the interactive and dynamic nature of real-world human–agent collaboration. To bridge this gap, τ-bench (Yao et al., 2024) simulates realistic multi-turn dialogues between an agent and a user simulator, incorporating domain-specific API tools and policy constraints. Building on this, τ^2-bench (Barres et al., 2025) extends the paradigm to dual-control environments, where both the agent and the user can act within a shared environment. In parallel, UserBench (Qian et al., 2025a) provides a more realistic setting in which agents must proactively clarify underspecified and incrementally revealed user goals. VitaBench (He et al., 2025) further challenges agents with a complex life-service simulation environment. On the methods side, several works train agents to proactively interact with users (Zhang et al., 2025; Chen et al., 2024), and UserRL (Qian et al., 2025b) leverages reinforcement learning to explicitly optimize robustness in multi-turn user interactions. However, this line of work largely overlooks systematic management of long, tool-augmented interaction histories. Our framework complements it by providing a dynamic, intent-aware context-folding solution.

2.2 Context Managment and Compression

As a standard paradigm, ReAct (Yao et al., 2022b) uses no explicit context management: it appends all thoughts, actions, and observations into a continually growing history, leading to context explosion as interaction continues. To mitigate this, one line of work adds external memory (e.g., A-Mem (Xu et al., 2025), Mem-OS (Li et al., 2025)), often at the cost of higher system complexity. Another line compresses the working context (Wu et al., 2025; Chen et al., 2025; Ye et al., 2025; Zhou et al., 2025c; Sun et al., 2025). ReSum (Wu et al., 2025), AgentFold (Ye et al., 2025), and IterResearch (Chen et al., 2025) iteratively summarize prior context to

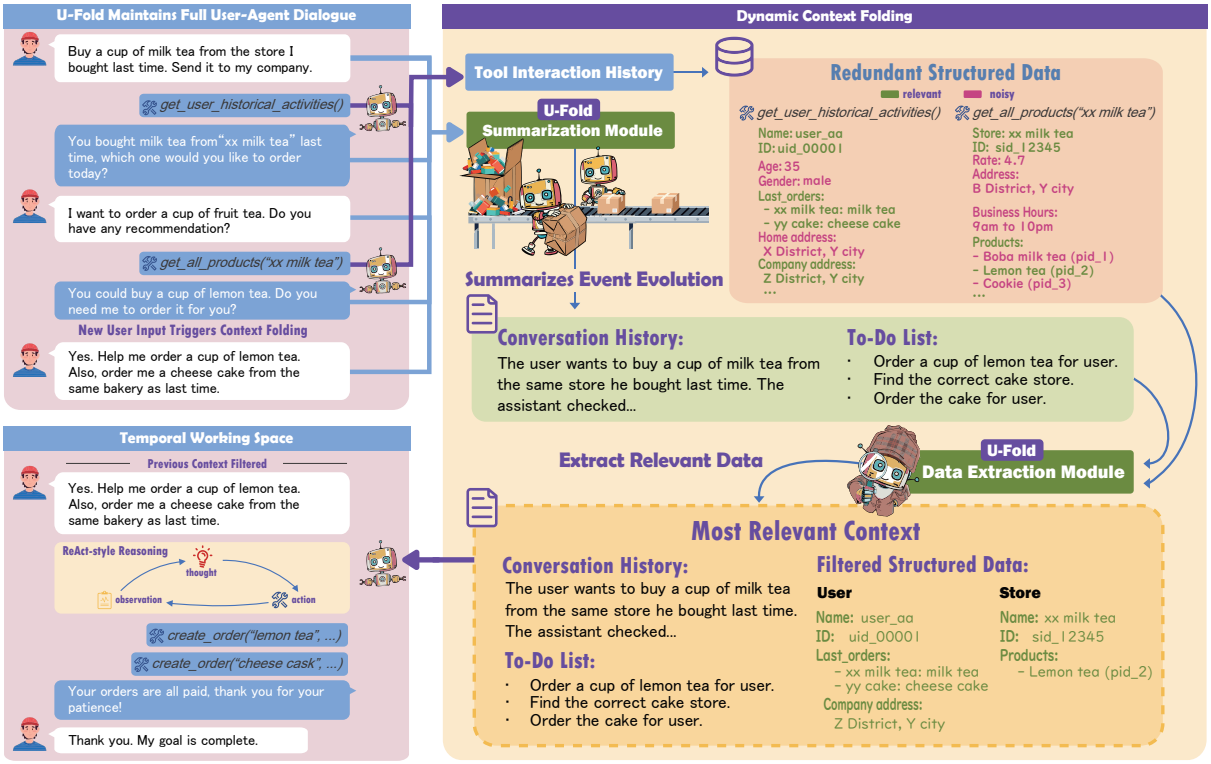

Figure 2: Overview of the U-Fold pipeline. U-Fold consists of two core components for dynamic context folding: (i) a Summarization Module that tracks the evolution of the conversation and maintains an explicit to-do list, and (ii) a Data Extraction Module that filters redundant structured tool outputs and retains only task-relevant information.

fit token budgets, while Sun et al. (2025) branches temporary contexts for subtasks and later merges them into the main trajectory. However, these compression methods largely target single-goal tasks and do not explicitly support user-centric settings with evolving intent. In contrast, U-Fold offers dynamic, intent-aware folding for user-centric scenarios, maintaining a compact context while handling long tool-augmented histories and shifting user goals.

3 Methodology

3.1 User-Centric LLM-Based Agent Paradigm

We model the behavior of an LLM-based agent as a Partially Observable Markov Decision Process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{O}$ denotes the observation space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denotes the transition function, and $\mathcal{R}$ denotes the reward function. In the user-centric setting, the agent interacts with both the database and the user via a suite of API tools as well as its final response, i.e., $\mathcal{A} = \mathcal{A}_{tool} \cup \mathcal{A}_{resp}$. Accordingly, the state space $\mathcal{S}$ comprises the database state and the user state, $\mathcal{S} = \mathcal{S}_{db} \otimes \mathcal{S}_{user}$, and the observation space $\mathcal{O}$ consists of tool-call outputs and user feedback, $\mathcal{O} = \mathcal{O}_{db} \cup \mathcal{O}_{user}$.

At the outset, the user initiates the conversation by issuing a query q_1. The agent then addresses the query by repeatedly executing cycles of *Thought*, *Action*, and *Observation* under the ReAct framework (Yao et al., 2022b). At cycle t, the agent generates a reasoning trace τ_1^t and an action a_1^t conditioned on the preceding context:

$$(\tau_1^t, a_1^t) \sim \pi_\theta(\cdot \mid q_1, \tau_1^1, a_1^1, o_1^1, \ldots, \tau_1^{t-1}, a_1^{t-1}, o_1^{t-1}),$$

where π_θ denotes the agent policy model and $\{o_1^1, o_1^2, \ldots, o_1^{t-1}\}$ denotes the tool-call feedback observed at each cycle in the first turn. The turn terminates when the agent chooses to produce a final response. The resulting sequence of agent decisions can be represented as a trajectory T_1:

$$T_1 = (\tau_1^1, a_1^1, o_1^1, \ldots, \tau_1^t, a_1^t)$$

In the user-centric setting, the user may issue follow-up queries conditioned on the preceding dialogue. At turn i, the user query q_i is generated as

$$q_i \sim \pi_\phi(\cdot \mid q_1, T_1, q_2, T_2, \ldots, q_{i-1}, T_{i-1}) \quad (1)$$

Here, π_ϕ denotes the user policy that captures user behavior. Likewise, for the agent at turn i and inner

cycle t, the generation of its reasoning trace τ_i^t and action a_i^t can be written as

$$(\tau_i^t, a_i^t) \sim \pi_\theta(\cdot \mid q_1, T_1, \ldots, q_i, \tau_i^1, \ldots, \tau_i^{t-1}, a_i^{t-1}, o_i^{t-1}) \quad (2)$$

3.2 U-Fold: User-Centric Dynamic Context Folding

Existing context folding methods rely on static, lossy compression, which degrades performance in user-centric scenarios with evolving user intents and fine-grained data requirements. To address this limitation, we propose U-Fold, a dynamic context folding framework. As depicted in Figure 2, U-Fold maintains a complete dialogue and tool call information throughout the interaction, while the agent always acts on a compressed view extracted from this full history at each step. U-Fold is triggered on every new user input and decomposes context management into two coordinated components: a ***Conversation Summarization module*** that maintains an up-to-date, user-centric abstraction of the interaction, and a ***Dynamic Data Extraction module*** that selectively retrieves task-relevant information from the tool-call history without discarding potentially important details.

3.2.1 Conversation Summarization

The Conversation Summarization module is an LLM-based summarizer that compresses the user-agent dialogue into a compact yet informative representation. Given the conversation history, it produces a structured summary that goes beyond recording the agent's past actions by explicitly tracking how the user's goals, constraints, and preferences evolve over time. In addition to the textual summary, the module generates an explicit to-do list that enumerates pending and newly introduced objectives. Each to-do item corresponds to a concrete subtask (e.g., "verify X," "compare Y," or "book Z under constraint C"), thereby providing an actionable interface between dialogue summarization and downstream planning. Formally, the conversation history C_i and the resulting summary $\mathcal{M}_i$ are defined as:

$$\mathcal{H}_i = (\tau_i^1, a_i^1, \ldots, \tau_i^t, a_i^t) \quad (3)$$

$$C_i = (q_1, \mathcal{H}_1, q_2, \mathcal{H}_2, \ldots, q_{i-1}, \mathcal{H}_{i-1}, q_i) \quad (4)$$

$$\mathcal{M}_i \sim \pi_{\theta_c}(\cdot \mid C_i) \quad (5)$$

Here π_{θ_c} denotes the policy model of the LLM-based summarizer. By consulting the summary and to-do list, the agent can recover the salient interaction history and infer the current conversational state without revisiting the raw dialogue. This design helps the agent stay aligned with evolving user needs while keeping the context representation concise and tractable.

3.2.2 Dynamic Data Extraction

Tool calls often return exhaustive user- and task-related information (e.g., full user profiles, product catalogs or complete database records), even though only a small subset is pertinent to the user's current needs. Naively feeding all returned content to the agent quickly bloats the context with noise and distractors. To address this issue, we introduce the Dynamic Data Extraction module, an LLM-based data selector. Conditioned on the history summary and to-do list produced by the Conversation Summarization module, it reviews the full logs of past tool calls and dynamically extracts only the information that is currently useful for resolving the user's pending goals. Irrelevant or redundant fields (e.g., static identifiers, unused attributes, or obsolete options) are pruned, yielding a compact yet semantically sufficient view of the data. The generation of dynamic data $\mathcal{D}_i$ is formally defined as

$$\mathcal{D}_i \sim \pi_{\theta_d}(\cdot \mid \mathcal{M}_i, T_1, T_2, \ldots, T_{i-1}), \quad (6)$$

where π_{θ_d} denotes the policy model of the data selector.

3.2.3 Discussions

Under the U-Fold framework, long and heterogeneous interaction traces are transformed into a highly condensed yet information-complete context. At each turn, the agent conditions on a compact bundle comprising (i) a user-centric, globally consistent summary of the dialogue trajectory and (ii) a dynamically curated subset of tool-returned data that is strictly relevant to the current goals:

$$(\tau_i^t, a_i^t) \sim \pi_\theta(\cdot \mid \mathcal{M}_i, \mathcal{D}_i, q_i, \tau_i^1, \ldots, \tau_i^{t-1}, a_i^{t-1}, o_i^{t-1}) \quad (7)$$

This design provides two advantages over standard context folding. First, it enforces *parsimony*: redundant details are aggressively removed, preventing context bloat (see Figure 3 (a)). Second, it maintains *sufficiency*: the dialogue summary and extracted data together retain all information required for correct reasoning and tool use (see Figure 3 (b) and 4). Because both the summary and the extracted data are recomputed after each new

Model	Framework	tau-bench		tau2-bench			VitaBench				Improvement
		Retail	Airline	Retail	Airline	Telecom	Delivery	In-store	OTA	Cross-Scenarios	
Qwen3-4B	ReAct	**22.6**	32.0	21.9	34.0	19.3	9.3	5.0	0.0	0.0	↑1.1
	IterResearch	19.1	26.0	14.1	22.0	**24.6**	**13.0**	7.0	0.0	0.0	↑3.7
	ReSum	20.0	30.0	20.2	30.0	18.4	5.0	2.0	0.0	0.0	↑3.7
	U-Fold (Ours)	21.7	**34.0**	**22.8**	**36.0**	19.3	10.0	**8.0**	0.0	**2.0**	
Qwen3-Thinking-30B-A3B	ReAct	**67.0**	**46.0**	62.3	52.0	30.7	33.3	43.5	11.7	15.0	↑1.5
	IterResearch	43.5	42.0	34.2	44.0	20.2	24.5	41.8	12.0	10.8	↑11.3
	ReSum	53.9	40.0	51.8	48.0	**33.3**	32.0	41.3	9.3	9.3	↑6.2
	U-Fold (Ours)	65.2	44.0	**70.2**	**54.0**	28.9	**35.0**	**44.0**	**13.8**	**19.5**	
DeepSeek-V3.2-Exp	ReAct	70.0	46.0	63.2	**70.0**	34.2	48.5	56.3	17.5	20.0	↑3.4
	IterResearch	24.3	42.0	7.0	36.0	18.4	29.5	28.8	15.0	11.8	↑27.0
	ReSum	55.7	46.0	64.0	60.0	22.9	31.5	20.3	5.0	8.3	↑15.8
	U-Fold (Ours)	**74.8**	**54.0**	**64.9**	68.0	**37.7**	**56.8**	**58.0**	**21.0**	**21.0**	
GPT-4.1	ReAct	72.2	60.0	70.2	64.0	43.9	45.5	55.8	29.0	17.3	↑2.5
	IterResearch	65.2	50.0	62.5	55.0	33.3	26.0	38.5	16.0	8.8	↑13.9
	ReSum	46.1	46.0	54.2	58.5	28.1	40.0	46.5	25.8	21.3	↑12.7
	U-Fold (Ours)	**73.0**	**61.5**	**70.8**	**65.5**	**44.7**	**51.3**	**58.5**	**33.0**	**22.3**	
Claude-4.5-Sonnet	ReAct	85.2	**56.0**	74.1	66.5	44.7	55.8	56.0	45.5	35.5	↑2.8
	IterResearch	76.5	52.0	71.1	63.0	29.0	50.8	59.8	30.0	27.0	↑9.5
	ReSum	69.6	54.0	70.4	65.0	42.1	53.8	55.0	34.0	35.3	↑7.3
	U-Fold (Ours)	**86.1**	54.0	**77.2**	**72.0**	**45.6**	**59.0**	**64.0**	**48.8**	**38.0**	

Table 1: **Avg@4 results** on τ-bench, τ^2-Bench, and VitaBench. The best results are highlighted in **bold**, and the second-best results are underlined. The "Improvement" column reports the average performance gain of U-Fold (ours) over each baseline across all domain tasks. We re-implemented and ran all baselines in our unified experimental setup.

user input, U-Fold's compressed context continuously tracks the user's evolving intent throughout the interaction. Consequently, information utilization becomes both simple and efficient: the agent can focus on addressing the current user needs without being distracted by irrelevant historical traces, while still benefiting from the full interaction history maintained in the background.

4 Experiments

4.1 Experimental Setup

Benchmarks. We conduct experiments on several challenging and widely used user-centric benchmarks: τ-bench (Yao et al., 2024), τ^2-Bench (Barres et al., 2025), and VitaBench (He et al., 2025). All three benchmarks feature multi-turn interactions with gradually shifting user demands, requiring the agent to invoke tools appropriately to fulfill user requests.

Baselines. We compare U-Fold with the standard agentic paradigm ReAct (Yao et al., 2022b) as well as representative context-folding methods, including ReSum (Wu et al., 2025) and IterResearch (Chen et al., 2025). We evaluate these approaches using both closed-source LLMs (GPT-4.1 (OpenAI, 2025) and Claude-4.5-Sonnet (Anthropic, 2025)) and open-source LLMs (DeepSeek-V3.2-Exp (Liu et al., 2025), Qwen3-Thinking-30B-A3B, and Qwen3-4B (Yang et al., 2025)).

Implementation Details. For U-Fold, we use the same backbone model for both conversation summarization and dynamic data extraction as for the main agent. We use GPT-4.1 as the user simulator. For VitaBench, we use GPT-4.1 as the LLM-based evaluator. During evaluation, we set the temperature to 0.0 for both the user simulator and the agent. For the main results, we report the average reward over four independent runs (Avg@4). For fair comparison, we re-implemented and ran all baselines in our unified experimental setup.

4.2 Main Results

Table 1 displays the comprehensive experimental results. Across all backbones and benchmarks, U-Fold achieves the best performance in nearly every setting and consistently outperforms existing context folding methods, while also rivaling or surpassing the full-context ReAct baseline.

Information-Preserving Compression Beats Full-Context Access. Although ReAct has access to the full dialogue and tool-call history, U-Fold often achieves better performance while operating on a compressed context. Compared with benchmarks with short contexts (τ-bench and τ^2-bench), U-Fold's advantage is more pronounced on VitaBench, where conversations are longer, tool outputs are substantially more verbose, and user intent is more complex. In particular, U-Fold consistently outperforms

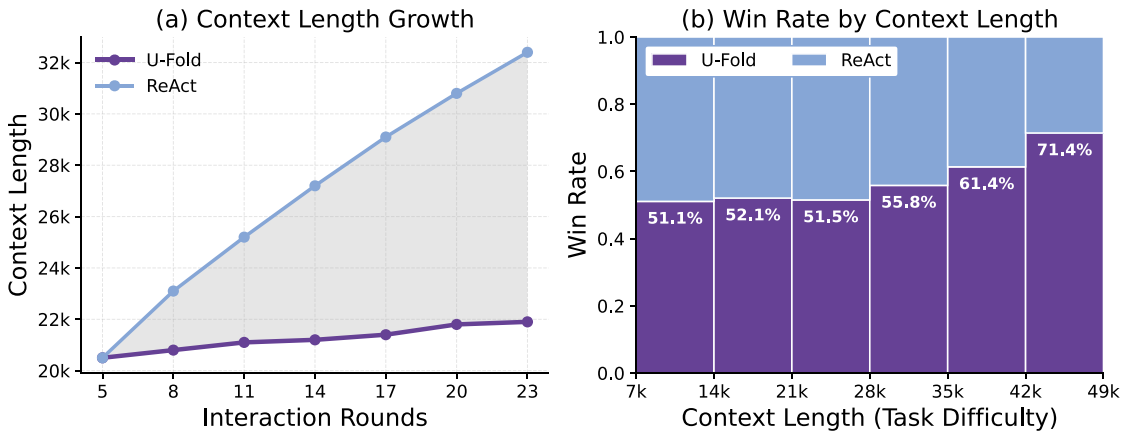

Figure 3: Context folding analysis of U-Fold against ReAct. (a) U-Fold substantially slows context length growth over interaction rounds while preserving task performance. (b) U-Fold win rate over ReAct across bins of final context length (a proxy for task difficulty), where final context length is ReAct's context size at task completion. For each length bin, the win rate is the ratio between the number of tasks solved by U-Fold and those solved by ReAct. U-Fold's relative advantage increases as final context length grows.

ReAct across domains such as *Delivery*, *OTA*, and *Cross-Scenarios* (e.g., *DeepSeek-V3.2-Exp Delivery* 56.8 vs. 48.5; *GPT-4.1 Cross-Scenarios* 22.3 vs. 17.3; *Claude-4.5-Sonnet OTA* 48.8 vs. 45.5), indicating that aggressive yet information-preserving compression is especially beneficial in long-horizon, noisy tool-use settings.

The magnitude of the gains also depends on backbone capacity. With a small backbone (Qwen3-4B), improvements over ReAct are modest. We attribute this to the limited ability of smaller models to serve as effective conversation summarizers and data extractors, which constrains the quality of the resulting compressed context (see Section 4.6.2). In contrast, stronger models (DeepSeek-V3.2-Exp, GPT-4.1, and Claude-4.5-Sonnet) yield larger and more consistent improvements, as the U-Fold modules can produce higher-fidelity summaries and more accurate data selection.

Dynamic, Intent-Aware Folding Minimizes Information Loss. Compared with IterResearch and ReSum, U-Fold yields substantial and consistent gains across all settings (with improvements of up to 27.0 over IterResearch and 15.8 over ReSum) by addressing two key limitations of static folding methods: (i) the irreversible loss of fine-grained constraints and intermediate facts (Figure 1), and (ii) insufficient summary content when user intent shifts over time (Figure 4). U-Fold mitigates these issues by retaining the full interaction and tool-call logs and dynamically extracting only the information relevant to the current turn. This design reduces redundant tool calls (Section 4.3) and lowers failure

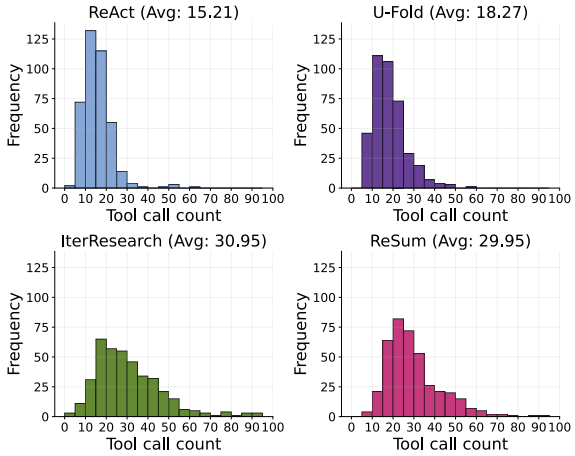

Figure 4: Distribution of tool-call counts under different agentic strategies. Static context folding methods repeatedly invoke tools to recover information lost during the context compression.

rates due to missed user constraints (Section 4.4), indicating that U-Fold better preserves critical information while still providing the agent with a compact, non-redundant context.

4.3 Context Folding Analysis

To assess the context-folding efficiency of U-Fold, we track context-length growth as the number of interaction rounds increases. As shown in Figure 3(a), U-Fold substantially slows context growth compared with ReAct (Yao et al., 2022b), indicating that our pipeline effectively compresses the working context as the dialogue progresses. Figure 3(b) further highlights U-Fold's advantage under severe context inflation. Here, the x-axis (*Context Length*)

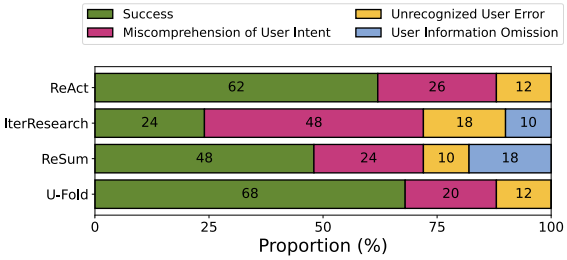

Figure 5: Error analysis on 50 randomly sampled tasks. We report the proportion of successes and three types of failures. U-Fold substantially reduces all error types, especially errors caused by missing user information.

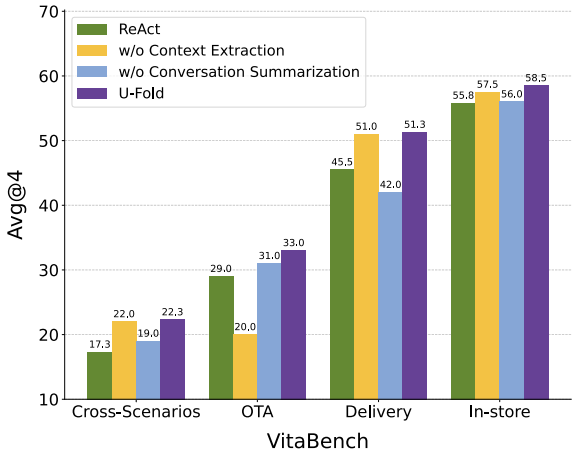

Figure 6: Ablation study of U-Fold. Both modules contribute to the overall gains, and removing either component degrades performance, confirming their complementary roles in effective context folding.

denotes the final context length of ReAct upon task completion, which serves as a proxy for task difficulty. For each interval, we compute the win rate as the ratio of tasks solved by U-Fold to those solved by ReAct. As context length increases, U-Fold's win rate rises steadily, suggesting that our method is particularly beneficial for harder long-context tasks and underscoring the importance of effective context folding.

Figure 4 compares the distribution of tool-call counts across different agentic strategies. Conventional context-folding methods (IterResearch (Chen et al., 2025) and ReSum (Wu et al., 2025)) periodically summarize and discard the raw history. As user intent evolves, crucial details may no longer appear in the current summary. This information loss forces the agent to repeatedly invoke the same tools to recover missing facts, resulting in substantially more tool calls. In contrast, U-Fold retains the full history and dynamically extracts intent-relevant data whenever the user issues a new query, ensuring that critical information remains accessible in the compressed context. Consequently, U-Fold achieves higher task success while requiring far fewer redundant tool invocations.

4.4 Error Analysis

To better understand U-Fold's behavior on user-centric tasks, we conduct a fine-grained error analysis on 50 randomly sampled VitaBench (He et al., 2025) tasks. For each method, we manually categorize the failed tasks into three types: (1) ***Miscomprehension of User Intent***, where the agent drifts from the user's evolving goals; (2) ***Omission of Critical User Information***, where key constraints or preferences are not reflected in the agent's context; and (3) ***Unrecognized User Errors***, where the agent fails to detect that the user's inputs are incomplete or inconsistent. As shown in Figure 5, U-Fold achieves the highest success rate and substantially reduces all three error types compared with baselines, with the largest gain in mitigating omissions of critical user information. This aligns with our design: by maintaining full history and dynamically extracting intent-relevant details, U-Fold keeps user constraints explicitly represented in the working context, enabling more faithful intent tracking and more reliable behavior in user-centric, multi-turn interactions.

4.5 Ablation Study

We conduct an ablation study on VitaBench (He et al., 2025) to quantify the contribution of each U-Fold component. The *w/o Context Extraction* variant retains the full tool-call outputs without filtering, while still using conversation summaries and to-do lists. The *w/o Conversation Summarization* variant, in contrast, keeps the raw dialogue while extracting only the relevant tool logs. As shown in Figure 6, removing either module degrades performance relative to full U-Fold. The drop is pronounced for *w/o Context Extraction* on the *OTA* domain: *OTA* involves many heterogeneous tools (e.g., tickets, attractions, and transportation options), and without dynamic data extraction the model is exposed to numerous irrelevant candidates, making planning more difficult. Conversely, *w/o Conversation Summarization* performs poorly on *Delivery*, where success depends on tracking user-provided addresses, time windows, and other implicit constraints across multiple turns. With-

Framework	In-store		Cross-Scenarios	
	Normal	Hard	Normal	Hard
ReAct	43.5	11.0	15.0	4.0
U-Fold	**44.0**	**27.0**	**19.5**	**12.0**

Table 2: Performance comparison (Avg@4) between ReAct and U-Fold under the standard and hard settings on VitaBench.

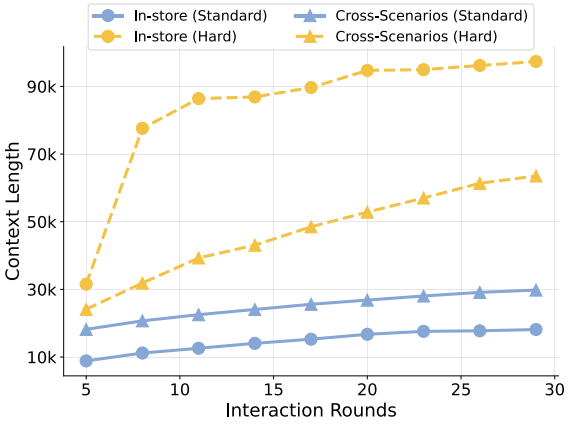

Figure 7: Context-length growth (under the ReAct framework) in the standard and hard settings on VitaBench.

out explicit summarization, the agent often fails to recover these latent requirements from the raw history. Overall, these results suggest that **Conversation Summarization** and **Dynamic Context Extraction** are both necessary and complementary to U-Fold's effectiveness.

4.6 Further Analysis

4.6.1 Evaluation on More Challenging Benchmark

To evaluate U-Fold under more severe context inflation, we construct a harder version of VitaBench (He et al., 2025) for the *In-store* and *Cross-Scenarios* domains using Qwen3-Thinking-30B-A3B (Yang et al., 2025) as the backbone. In this hard setting, tool calls return redundant information as noisy distractors, causing the context length to grow much faster than in the standard setting (Figure 7). This setup better reflects real deployments, where tool APIs may expose verbose and noisy payloads.

Table 2 compares ReAct (Yao et al., 2022b) and U-Fold across both difficulty levels. Under the hard setting, U-Fold's advantage becomes substantially larger: performance increases from 11.0 to 27.0 on *In-store* and from 4.0 to 12.0 on *Cross-Scenarios*, despite the much faster context growth induced by redundant tool outputs. These results suggest that U-Fold is particularly effective in the most demanding long-context, user-centric scenarios.

Framework	VitaBench			
	Delivery	In-store	OTA	Cross-Scenarios
ReAct	9.3	5.0	0.0	0.0
U-Fold	10.0	8.0	0.0	**2.0**
U-Fold + Better Folder	**17.0**	**10.0**	**3.0**	1.0

Table 3: Capability transfer via U-Fold: Qwen3-4B as the main agent, with U-Fold using either the same backbone or a stronger folder (GPT-4.1). Results are reported as Avg@4 scores.

4.6.2 Capability Transferring through Context Folding

We further examine whether a stronger LLM can act as an informative folder for a weaker agent. In this setting (*U-Fold + Better Folder*), the more capable folder (GPT-4.1 (OpenAI, 2025)) generates higher-quality conversation summaries and dynamically extracted tool contexts, which are then consumed by the weaker agent (Qwen3-4B (Yang et al., 2025)). As shown in Table 3, this configuration improves over both ReAct (Yao et al., 2022b) and U-Fold, suggesting that capabilities of a large model can be transferred via better context shaping rather than end-to-end inference. Practically, this points to a cost-effective deployment pattern: using a heavyweight LLM sparingly as a folding service to boost cheaper small agents in long-horizon, tool-intensive interactions.

5 Conclusion

In this work, we study context-folding strategies for user-centric, tool-augmented settings and reveal key limitations of existing approaches: static summaries cannot reliably track evolving user intent, often discard critical constraints, and consequently induce redundant tool calls or erroneous actions. To address these issues, we propose U-Fold, a dynamic context-folding framework that (i) continuously summarizes the evolving conversation and maintains an explicit to-do list, and (ii) adaptively extracts task-relevant structured information from the full tool-call history. Extensive experiments on challenging user-centric benchmarks, together with comprehensive analyses, show that U-Fold consistently outperforms both ReAct (Yao et al., 2022b) and prior folding baselines, especially on long and noisy tasks. These findings suggest that future work on context management should move beyond single-intent summarization toward intent-aware, structure-preserving folding.

Limitations

U-Fold delivers substantial gains in user-centric, tool-augmented scenarios, but there remains room for improvement. First, U-Fold currently performs summarization and data extraction at every user turn, rather than explicitly detecting whether user intent has changed. Learning a policy to decide when to trigger folding could further reduce overhead and improve robustness. Second, existing benchmarks still fall short of the complexity of real-world multi-session, multi-user systems. Building more challenging, large-scale benchmarks with richer tool ecosystems and longer, more interdependent tasks is an important direction for future work.

References

Anthropic. 2025. Introducing claude sonnet 4.5.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Wayne Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, and 1 others. 2025. Iterresearch: Rethinking long-horizon agents via markovian state reconstruction. *arXiv preprint arXiv:2511.07327*.

Maximillian Chen, Ruoxi Sun, Tomas Pfister, and Sercan Ö Arık. 2024. Learning to clarify: Multi-turn conversations with action-based contrastive self-training. *arXiv preprint arXiv:2406.00222*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. 2025. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *arXiv preprint arXiv:2509.26490*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Zhiyu Li, Shichao Song, Chenyang Xi, Hanyu Wang, Chen Tang, Simin Niu, Ding Chen, Jiawei Yang, Chunyu Li, Qingchen Yu, and 1 others. 2025. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*.

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*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2025. Introducing gpt-4.1 in the api.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, and 1 others. 2025a. Userbench: An interactive gym environment for user-centric agents. *arXiv preprint arXiv:2507.22034*.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jielin Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, and 1 others. 2025b. Userrl: Training interactive user-centric agent via reinforcement learning. *arXiv preprint arXiv:2509.19736*.

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. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

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*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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*.

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*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. In *The eleventh 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, and 1 others. 2025. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, and 1 others. 2025. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, and 1 others. 2025a. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*.

Yuyang Zhou, Jin Su, Jiawei Zhang, Wangyang Hu, Tianli Tao, Guanqi Li, Xibin Zhou, Li Fan, and Fajie Yuan. 2025b. Prime: A multi-agent environment for orchestrating dynamic computational workflows in protein engineerings. *bioRxiv*, pages 2025–09.

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

A Algorithm Pseudo-Code

Algorithm 1: U-Fold: User-Centric Dynamic Context Folding

Input: New user query q_t, conversation history $C_t = (q_1, \mathcal{H}_1, \ldots, q_{t-1}, \mathcal{H}_{t-1}, q_t)$, tool history $\{T_1, \ldots, T_{t-1}\}$, agent policy π_θ, summarizer π_{θ_c}, data extractor π_{θ_d}, tool set $\mathcal{A}_{tool}$, response action $\mathcal{A}_{resp}$

Output: Agent trajectory T_t

$\mathcal{M}_t \sim \pi_{\theta_c}(\cdot \mid C_t)$ // Conversation summarization, Eq. (5)

$\mathcal{D}_t \sim \pi_{\theta_d}(\cdot \mid \mathcal{M}_t, T_1, \ldots, T_{t-1})$ // Dynamic data extraction, Eq. (6)

// Run ReAct-style inner loop conditioned on folded context

$T_t \leftarrow []$;

$i \leftarrow 1$;

while *final response not produced* **do**

  $(\tau_t^i, a_t^i) \sim \pi_\theta(\cdot \mid \mathcal{M}_t, \mathcal{D}_t, q_t, \tau_t^{1:i-1}, a_t^{1:i-1}, o_t^{1:i-1})$ // Eq. (7)

  if $a_t^i \in \mathcal{A}_{tool}$ **then**

    $o_t^i \leftarrow \text{ExecuteTool}(a_t^i)$

    $T_t \leftarrow T_t + [(\tau_t^i, a_t^i, o_t^i)]$

  else

    // $a_t^i \in \mathcal{A}_{resp}$ is the final natural-language reply

    $T_t \leftarrow T_t + [(\tau_t^i, a_t^i)]$

    break

  $i \leftarrow i + 1$;

return T_t

B U-Fold Prompts

In this section, we provide the full prompts used in U-Fold for all two components: (1) Conversation Summarization, which maintains the evolving summary and to-do list; (2) Dynamic Data Extraction, which filters and structures tool outputs into a compact context. Additionally, we provide the prompt for the Main Agent, which performs ReAct-style reasoning and tool invocation based on the folded context.

B.1 Conversation Summarization

The detailed prompt for Conversation Summarization is as follows:

Prompt for Conversation Summarization

```
You are a dialogue history condenser.
You are given:
- A single, highly condensed,
high-information summary that has
been generated from you based on the
conversation history.
===
PUT_HISTORY_HERE
===
- The new conversation history to tell
you what has happened between the
user and the assistant.
===
PUT_CONVERSATION_HERE
===
Your task:
Based on above information, produce a
single, highly condensed,
high-information summary that:
- Preserves all essential facts,
constraints, decisions, and
assumptions.
- Clearly reflects the chronological
order of events and requests.
- Makes the temporal and causal
relationships explicit (what happened
first, next, and as a result of what).
- Captures the user's goals,
questions, and changes of mind over
time.
- Captures the assistant's key
answers, outputs, plans, and action
steps (explicitly shows tool name if
the tool is used, shows digital
identifiers for any data).
- Includes any important time
references, deadlines, versions,
identifiers, or states ("now",
"later", "step 2 finished", etc.).
- If the absolute time is clear,
specifically clarify it (day, month
etc.) and the relative time (tomorrow,
yesterday etc.).
- If the input context is in Chinese,
output in Chinese. If the input
```

```
context is in English, output in
English.
Formatting rules:
- Output only one plain text block.
- Do NOT include bullet points, lists,
headings, metadata, or commentary
about the task.
- Write as a compact narrative, in
chronological order, while remaining
as concise as possible without losing
critical information.
- Output the same language as the
input.
Finally, based on the above, output a
to-do list that the agent should do to
complete the user's goal. Only list
those actions have not been done yet:
To-do list (The task should follow
the execution order):
Step1. Task 1
Step2. Task 2
...
When you generate sub-tasks,
detailedly describe the task. Provide
all essential facts and constraints
to avoid any ambiguity. Don't
hallucinate anything not mentioned in
the conversation history.
```

B.2 Dynamic Data Extraction

The detailed prompt for Dynamic Data Extraction is as follows:

Prompt for Dynamic Data Extraction

```
You are a context-filtering agent.
Your goal is to select all useful
past information for the user query.
You are given:
- A list of tools that help you find
helpful information for tool use:
===
PUT_TOOLS_HERE
===
- A conversation history to tell you
what has happened between the user
and the assistant:
===
PUT_CONVERSATION_HERE
===
- A full interaction history with
thought-action-observation triples:
===
PUT_CONTEXT_HERE
===
# Your tasks
Based on the to-do list in
conversation history, From the entire
thought-action-observation triples,
select every OBSERVATION (and only
observations) lines that might help
answer the new user query, avoid
redundant tool calls, or support
reasoning.
Important rules:
- You may only extract from
**observations** in the full
interaction history with
thought-action-observation triples.
Do not use thoughts or actions.
- You must **not** output any original
observation text; only refer to it by
line numbers.
- The purpose is context
simplification by **line-range
selection**, not summarization or
rewriting of the original content.
- Prefer observation lines that:
- Contain facts, intermediate results,
or tool outputs relevant to the
query's constraints. - Help avoid
repeating the same tool call or
computation.
- Are likely to be reused for
reasoning or final answering.
- Irrelevant or weakly related
observations should be omitted.
# Output format
You should output a list of selected
context blocks. For each block, you
should output:
- Summary: Your concise summary in
your own words and how it is related
to the to-do list in conversation
history (don't extract unnecessary
information).
- Original: output the line range in
the format "Lines: <start>-<end>". If
you only need a single line N, output
"Lines: N-N".
```

- Facts: **ALL** concrete facts explicitly stated in this item (entities, values, states, time info, conditions): - For each key information, output its original text. Do NOT rewrite, paraphrase, or edit any extracted text. Every extracted block must be copied verbatim from the original history. Not a single character should be changed, added, or removed.
- Constraints: **ALL** limitations, rules, requirements that are explicitly stated or strictly logically implied (no speculation).
- Hint: Generate a detailed, direct guidance for the main task agent to solve the to-do list based on above information. Additionally follow these general principles:
- When the user refers to past activities, orders, reservations, or interactions (e.g. "the last restaurant", "the hotel I booked before"), do not guess which specific entity they mean.
- Always use tools to retrieve concrete historical records and identify the correct entity by its system identifier (ID or similar). Point out explicitly which tool can be used.
- Never operate on or recommend an entity based only on its name or vague description. The main task agent must first obtain and confirm the entity's unique system identifier.
- All references to days or times (today, tomorrow, next Monday, this weekend, etc.) must be tied to an explicit absolute date and time.
- For any booking, scheduling, or time-sensitive action, the requested time must lie within the resource's valid time window (e.g. opening hours, service availability, active period).
- If the available information is not enough to determine whether the requested time is valid or feasible, the main task agent must ask the user for clarification instead of assuming.
- For any operation that requires a quantity or capacity (number of items, tickets, units, rooms, seats, etc.), the main task agent must base it on the user's explicit request or clearly stated context.
- When the quantity or configuration is not specified and cannot be safely inferred, the main task agent must ask the user directly and must not make up defaults.
- Do not try to change structured attributes (quantity, type, specification, options) indirectly via free-text notes or comments. Structured attributes must be changed using proper fields and supported operations.
- For tasks that involve reordering, modifying, or replacing an existing item, order, or reservation, the main task agent must first retrieve the previous record using tools.
- To modify such records, follow the system's lifecycle: retrieve → cancel or update as allowed → create a new record if needed. Never assume that the state has changed without confirmation from the tools.
- When the user asks to "order the same as last time" or to "repeat a previous purchase", the main task agent should reuse the concrete identifiers and parameters from the previous record instead of guessing.
- If a place, venue, or address cannot be confidently identified from the provided text, the main task agent must use appropriate tools (e.g., geocoding, map search) to get precise location data (coordinates or canonical address).
- For travel or transport planning, if no direct option is available, the main task agent is allowed to consider indirect routes or nearby locations, but always based on actual tool results, not on speculation.
- When critical information required to complete a task is missing (e.g.,

```
date, time, location, number of
participants, type of ticket), the
main task agent must ask the user
explicitly rather than infer or
assume values.
- When asking the user questions,
avoid leading or suggestive options
that could misdirect the user. Ask
neutral, direct questions that do not
bias the user toward a particular
choice.
- If the interaction history or tool
results clearly indicate that the
user has omitted an essential step
(such as purchasing a required ticket,
confirming a payment, or finalizing a
booking), the main task agent should
proactively remind the user and offer
to complete that step.
- When entities are described by
structured fields or tags, treat them
as the ground truth about what exists
or is supported.
- Any ability, attribute, or feature
not listed in the structured fields
should be assumed absent; any
attribute listed should be treated as
present.
- When answering user questions about
those entities, rely directly on the
structured fields instead of
speculating from general knowledge or
typical behavior.
- For any task involving delivery,
travel, or time-to-completion, the
main task agent must distinguish
between start/dispatch time and
arrival/completion time.
- Before scheduling a start time, the
main task agent should determine the
expected duration or time-to-arrival
using tools, then choose a start time
that satisfies the user's deadlines
or constraints.
- If the user specifies a deadline or
required arrival time, the main task
agent must explicitly check whether
the expected arrival time meets that
requirement; if not, it should
explain the conflict and propose
alternatives.
```

```
If nothing is relevant, return an
empty output.
If the input context is in Chinese,
output in Chinese. If the input
context is in English, output in
English.
```

B.3 Main Agent

The prompt for the Main Agent is as follows:

Prompt for Main Agent

```
# Absolute System Rules
Answer the following questions as
best you can. You have access to the
following tools:
===
PUT_TOOLS_HERE
===
# Selected Context from Interaction
History
The most relevant parts of your past
interactions with the user have been
extracted for you below.
These snippets come directly from the
full interaction history and may
contain:
- the user's goals and constraints,
- previously discussed plans or
partial solutions,
- important facts, assumptions, or
tool results,
- any other information that may help
you decide your next actions.
You should carefully read and refer to
this selected context when analyzing
the current problem and planning your
next step. When relevant, rely on
this information instead of re-asking
the user for things that are already
known and calling the same tool again.
Here is the selected context:
===
PUT_SELECTED_CONTEXT_HERE
===
# Output Format
At each round, you should first
generate a thought block wrapped by
<inner> and </inner>:
<inner>
```

```
First repeat the constraints and
hints in the selected context. If the
steps in to-do list contradicts the
constraints, you should follow the
constraints.
Then analyze the problem and your
current state based on selected
context. Then select an available
tool to execute for next step.
</inner>
Then, you can choose to execute a
tool or generate a final answer. If
you choose to execute a tool, you
should generate an action block
wrapped by <action> and </action>:
<action>
"action": "Action name, should be one
of available tools.",
"parameters":
"xxx": "xxx",
"yyy": "yyy",
</action>
If you choose to generate a final
answer, you should generate a final
block wrapped by <final> and </final>:
<final>
the final answer to the original
input question
</final>
# Output Example
User: What is the capital of China?
<inner>
I need to search the capital of China.
</inner>
<action>
"action": "get_capital",
"parameters":
"country": "China"
</action>
<observation>
Beijing
</observation>
<inner>
I have searched the capital of China.
</inner>
<final>
Beijing is the capital of China.
</final>
# User Additional Instructions
Here are some additional instructions
from the user. You should follow
these instructions but when these
instructions contradicts the system
rules, you should follow the system
rules. Remember that you should
always follow the output format
described above.
===
PUT_USER_INSTRUCTIONS_HERE
===
Remember, at any time, you MUST
follow the Output Format described by
system!!!
```